Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERIA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

ANÁLISIS DE DISEÑO Y PROPUESTA DE MEJORA DEL "PIPELINE" DE UN PROCESADOR RISC-V

Autor: Alberto Díaz Tendero Quijorna

Tutor/es: Agustín Martínez Hellín

Co-tutor: Iván Gamino del Río

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERIA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

ANÁLISIS DE DISEÑO Y PROPUESTA DE MEJORA DEL "PIPELINE" DE UN PROCESADOR RISC-V

Autor: Alberto Díaz – Tendero Quijorna

Tutor/es: Agustín Martínez Hellín / Iván Gamino del Río

TRIBUNAL:

Presidente: Manuel Prieto Mateo

Vocal 1º: Rafael Rico López

Vocal 2º: Agustín Martínez Hellín

FECHA: 06/10/2021

ÍNDICE

ÍNDICE DE FIGURAS

RESUMEN

Partiendo de la patente "Un método y un dispositivo de procesamiento en instrucciones de programa e instrucciones de traza", la cual presenta un mecanismo para evitar el intrusismo de las instrucciones de traza, el SRG (*Space Research Group)* ha desarrollado un procesador basado en arquitectura RISC-V de 32 bits. Dicho procesador está diseñado con un *pipeline* de cinco etapas, el cual se ha modificado para conseguir una mayor eficiencia y así aumentar su frecuencia de funcionamiento.

ABSTRACT

Based on the "A methon and a parallel processing device of program instruction and trace instruction" patent, which presents a mechanism to avoid the intrusion of the trace instructions, the Space Research Group developed a 32 bits processor based on RISC-V architecture. This processor is designed using a five-stage *pipeline* which has been modified to achieve greater efficiency and thus increase its operating frequency.

PALABRAS CLAVE

RISC-V, VHDL, Pipeline, Slack Time, WCET, Branches

1. INTRODUCCIÓN

Este trabajo nace de la investigación que se ha llevado a cabo en el grupo de investigación espacial de la Universidad de Alcalá (en inglés, *Space Research Group - SRG*). El SRG está trabajando actualmente en un procesador con arquitectura RISC-V, el cual iría destinado a proyectos espaciales. Este tipo de sistemas enviados hacia el espacio, deben de ser sistemas que funcionen en tiempo real. Por lo que deben de cumplir ciertas especificaciones para que realicen su cometido de manera adecuada. Estos sistemas de tiempo real tienen que ser deterministas, ya que se necesita conocer la respuesta que tendrán éstos ante cualquier tipo de evento. Otra característica es que tienen que ser planificables, es decir, conocer el tiempo de respuesta que tenga el sistema ante algún evento. En computación, lo que interesa conocer es el peor caso de ejecución posible del sistema (en inglés, *Worst Case Execution Time)*, conocido por sus siglas, WCET, que es el tiempo máximo en el que una tarea puede ejecutarse.

Pues bien, este procesador está diseñado con una arquitectura RISC-V y parte de la patente "Un método y un dispositivo de procesamiento en instrucciones de programa e instrucciones de traza". El objetivo de esta patente es eliminar el intrusismo de las instrucciones de traza, consiguiéndolo al añadir para estas instrucciones una unidad en paralelo, es decir utilizando una ruta para instrucciones de traza y otra para instrucciones nominales. [1] De esta patente, se ha publicado un Trabajo Fin de Máster [2] seguido de una publicación en la revista *Electronics* [3] los cuales implementan este demostrador de la patente, con un procesador con arquitectura RISC-V. Este procesador ha sido diseñado con *pipeline*, es decir segmentando las diferentes etapas para que trabaje de manera más eficiente.

En este trabajo se ha realizado un estudio sobre las posibles mejoras que podría tener el procesador para su optimización. Se ha trabajado junto al diseñador del procesador, para el análisis y comprensión del sistema. Una vez que se ha comprendido el funcionamiento básico del procesador, comienza la búsqueda del camino con peor tiempo de ejecución y por último se propone la mejora y se implementa en el sistema, realizando las pruebas oportunas para su validación.

Este trabajo fin de grado se compone de dos partes diferenciadas, una base teórica y una base experimental. En la base teórica, se realiza un estudio sobre la patente, la arquitectura RISC-V y finalmente del procesador. En esta parte también se hace un estudio detallado sobre la herramienta *Vivado*, la cual se utiliza para realizar todos los análisis. Por otro lado, la segunda parte del trabajo se compone de la parte práctica de la mejora, es decir, se ha estudiado las posibles mejoras y se ha llevado a cabo su implementación, mediante diferentes análisis. Una vez realizada la mejora, se ha implementado y se han realizado las comprobaciones oportunas, probando finalmente el sistema en el microprocesador. El final del trabajo acaba con una conclusión y un posible trabajo futuro.

Aparte de esto, se ha adjuntado en este trabajo, todos los anexos que explican la instalación y puesta en marcha de la herramienta *Vivado*, para su utilización en este análisis, además del código que se ha utilizado para la mejora.

2. BASE TEÓRICA

En esta parte se explican los antecedentes del proyecto derivados de la patente "Un método y un dispositivo de procesamiento en paralelo de instrucciones de programa e instrucciones de traza", además de todos los conceptos teóricos que tienen relación con el procesador, la arquitectura utilizada, RISC-V, además de los conceptos necesarios sobre la herramienta Vivado para el diseño y análisis del sistema.

2.1. LA PATENTE: "UN MÉTODO Y UN DISPOSITIVO DE PROCESAMIENTO EN PARALELO DE INSTRUCCIONES DE PROGRAMA E INSTRUCCIONES DE TRAZA"

Como se ha comentado en la introducción, los sistemas de tiempo real que se utilizan, por ejemplo, en proyectos espaciales, necesitan cumplir una serie de requisitos principales para su correcto funcionamiento. Entre ellos se encuentra el de ser un sistema determinista, para así poder calcular de manera precisa el tiempo de respuesta que tiene el sistema ante eventos de cualquier tipo. A nivel de computación, se mide en estos sistemas el tiempo conocido como "el peor caso de ejecución posible", conocido por sus siglas en inglés como "WCET", *Worst Case Execution Time*.

Un sistema de tiempo real también necesita ser planificable, ya que es necesario una medida de tiempo eficaz de las tareas que desarrolla este sistema, para poder predecir su comportamiento. Así, los tiempos de ejecución de las tareas de un sistema deben ser planificables consiguiendo estar por debajo de un cierto límite. Una vez que no es posible aumentar la velocidad del procesamiento del sistema, se necesita entonces mejorar la precisión de la medida.

En este caso, el grupo de investigación del espacio (SRG) ha optado por introducir instrucciones de traza entre las tareas del sistema para poder medir de forma más precisa los tiempos de ejecución entre ellas. Esto presenta algunos inconvenientes, como, por ejemplo, el sistema se vuelve menos determinista, afectando a su planificabilidad o, por otro lado, afectando al tiempo máximo de ejecución del sistema, el cual aumentaría, ya que habría que añadir el tiempo de ejecución de las trazas al WCET. Para combatir este intrusismo producido por las instrucciones de traza el SRG ha desarrollado una patente llamada "*Un método y un dispositivo de procesamiento en paralelo de instrucciones de programa e instrucciones de traza*". [1]

Esta invención consiste en añadir una unidad paralela en el procesador dedicada a ejecutar las instrucciones de traza. Así las instrucciones de programa y las instrucciones de traza pueden ejecutarse en paralelo favoreciendo al tiempo de ejecución. El dispositivo de procesamiento para estas instrucciones está estructurado de la siguiente manera:

- Una etapa para la búsqueda de instrucciones. Esta etapa se compone de un módulo de cálculo de la dirección de la instrucción y un módulo de búsqueda de las instrucciones con doble puerto de lectura (esto más adelante se referirá como la etapa IF).

- Una etapa de decodificación duplicada (ID).

- Un pipeline-traza para el procesado de instrucciones de traza únicamente. Paralelo a este, se encuentra el mismo para instrucciones de no traza, compuesto por las etapas de ejecución (EX), memoria (MEM) y escritura (WB).

- Un registro de salida para la traza.

- El *pipeline* de traza está compuesto por una ruta de datos que se compone de un conjunto de multiplexores y un controlador de la ruta de datos, compuesto por entradas y salidas que controlan los multiplexores. En función del valor de las entradas, el valor de las salidas es enviado a los multiplexores de la ruta de datos, ejecutándose de forma sincronizada una instrucción de traza con la instrucción que le precede. Esta ejecución se realiza durante la última etapa del *pipeline* de traza.

La implementación de la patente consiste en este *pipeline* adicional de traza añadido al procesador de la arquitectura RISC-V destinado únicamente al procesamiento y extracción de instrucciones de traza. Por lo tanto, la implementación se compone de dos *pipelines*, uno "nominal" (o de instrucciones de programa) y otro de traza (para las instrucciones de traza).

El procesador resultante trabaja con dos instrucciones al mismo tiempo, por lo que es necesario un mecanismo de doble búsqueda de instrucciones (con memoria de doble puerto) y un mecanismo de enrutamiento de las instrucciones en su correspondiente *pipeline*. Como existen dos tipos de instrucciones (de traza y nominales), y las instrucciones se procesan de dos en dos, existen 4 combinaciones posibles: N-N, N-T, T-N, T-T. A su vez, se ha realizado una división entre par de instrucciones homogéneas (N-N, T-T) y heterogéneas (N-T, T-N). Con esta división, el procesador posee un enrutador de instrucciones que dispone de una máquina de estados capaz de controlar el paso de las instrucciones a ambos *pipelines* y de solicitar nuevas instrucciones. Esto se realiza para controlar el flujo de instrucciones, y que en el caso de que tengamos dos instrucciones homogéneas, se pueda realizar un mecanismo de parada para que se ejecute una de esas dos instrucciones, y después la otra, ya que las dos no pueden pasar por el mismo camino. [2] [3]

2.2. ARQUITECTURA RISC-V

El procesador con el que se ha trabajado en este proyecto está diseñado bajo una arquitectura RISC-V. Ésta, es una arquitectura formada por un conjunto de instrucciones predefinido (ISA: *Instruction Set Architecture*) de código abierto, es decir accesible a cualquiera que quiera trabajar con ella. Su nombre proviene de las siglas en inglés *Reduced Instruction Set Computer* y utiliza instrucciones de tamaño fijo, presentadas en un formato reducido. Además, solo las instrucciones de carga y almacenamiento pueden acceder a la memoria de datos.

Pues bien, el procesador desarrollado se basa en este tipo de arquitectura con unas instrucciones predefinidas en formato fijo. Estas instrucciones tienen un total de 32 bits. Hay seis tipos de formato de instrucciones:

- Tipo-R para operaciones entre registros;
- Tipo-I para inmediatos cortos y *loads*;
- Tipo-S para *stores*;
- Tipo-B para *branches*;
- Tipo-U para inmediatos largos; y
- Tipo-J para saltos incondicionales.

En la siguiente Figura 1 se pueden observar estos tipos de instrucciones y su formato.

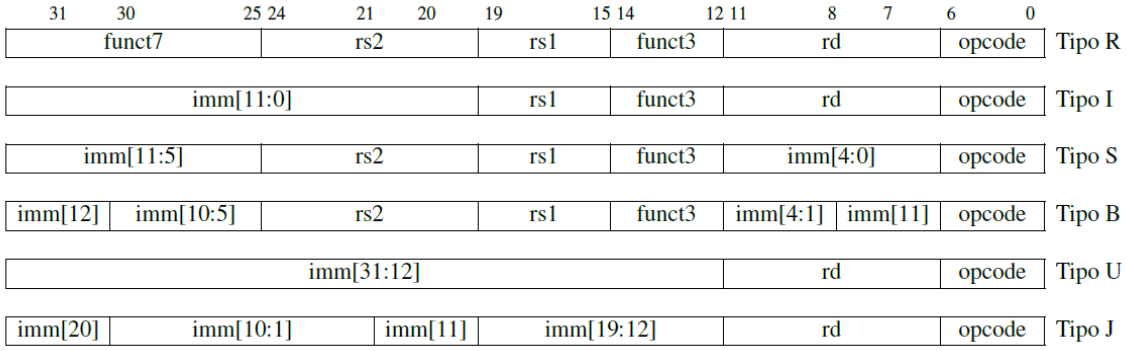


Figura 1: Formato de instrucciones RISC-V

Las instrucciones poseen un campo de 7 bits llamado *opcode*, el cual indica el código de operación a realizar por el procesador, es decir, dice al procesador qué acción tiene que realizar, como, por ejemplo, operaciones aritméticas o lógicas entre registros, saltos condicionales o incondicionales, guardar o escribir en un registro, etc.

Otro campo que se puede apreciar en casi todas las operaciones, menos para las instrucciones de tipo S y tipo B (*stores* y *branches*) es “rd”, que se corresponde con el registro destino, es decir, el registro sobre el que se va a guardar el resultado de la operación que se realice. En cuanto a otros campos relativos a este, se encuentran "rs1” y “rs2”, ambos indican los registros fuente, es decir, registros utilizados como operadores en la operación que indica la instrucción a ejecutar. El campo “imm”, indica inmediato, es

decir, es un valor fijo que se utiliza para realizar la operación, por ejemplo, para calcular desplazamientos, operaciones aritméticas o calcular direcciones de memoria.

Por último, se tienen los campos "func7" (bits del 25 al 31) y "func3" (bits del 12 al 14) de 7 y 3 bits, respectivamente. Estos campos indican la funcionalidad de la operación a realizar, por ejemplo, si se va a realizar una operación entre registros, de tipo R, los campos "func" mostrarán qué operación realizarán, es decir, si es un desplazamiento de bits a la derecha o a la izquierda, si es una suma o resta, etc.

En la Figura 2: conjunto de instrucciones RISC-V y su formatoFigura 2 se puede comprobar todos los campos mencionados anteriormente y los tipos de operaciones que se presentan dependiendo de la codificación de bits que se utilice. [4]

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	000	00000	1110011	I ecall
000000000001		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I csrrsi
csr		zimm	111	rd	1110011	I csrrci

Figura 2: conjunto de instrucciones RISC-V y su formato

2.3. PROCESADOR

El prototipo del procesador con el que se está trabajando proviene del paper "***A RISC-V Processor Design for Transparent Tracing***" [3]. Es una actual investigación sobre la implementación de un procesador con arquitectura RISC-V destinado a sistemas de tiempo real para proyectos espaciales, los cuales se están desarrollando en el **Grupo de Investigación Espacial de la UAH (SRG – Space Research Group)**. A su vez, el modelo que se ha escogido para el diseño del procesador es el proporcionado en la arquitectura de Patterson and Hennessy [5].

A este procesador se le ha añadido segmentación, ya que así se aumenta su eficiencia y velocidad, esenciales para el correcto funcionamiento de un sistema de tiempo real. La arquitectura RISC-V con la que trabajamos, posee 5 etapas definidas, de las cuales las 3 primeras son comunes a todas las instrucciones, y las 2 últimas dedicadas solo a algunas instrucciones específicas:

1. **INSTRUCTION FETCH (IF).** Búsqueda y extracción de las instrucciones alojadas en la memoria y su puesta en marcha en el flujo de datos.
2. **INSTRUCTION DECODE (ID).** Decodificación de las instrucciones.
3. **EXECUTE (EX).** Ejecución de las instrucciones.
4. **MEMORY (MEM).** Acceso a la memoria.
5. **WRITE-BACK (WB).** Guardado del resultado en un registro.

Así, este procesador está diseñado para introducir una instrucción desde la memoria al flujo de datos. Una vez en el flujo de datos, esa instrucción se decodifica para que pueda ser entendida por el compilador y se encamina hacia la etapa de ejecución, que es la etapa donde se procesa la ejecución de la instrucción. Estas tres primeras etapas, se deben de cumplir para todas las instrucciones, sean del tipo que sean. En cambio, para la etapa de acceso a memoria, solo se encaminan cierto tipo de instrucciones que necesitan acceder a la memoria para coger algún dato. Lo mismo ocurre con la última etapa de escritura en registro, utilizada solo para instrucciones que necesiten guardar un dato en un registro. Este es el funcionamiento a alto nivel del procesador.

El procesador RV32Xtrace, trabaja con registros de 32 bits, algunos de los cuales están reservados y otros que son de propósito general. En este diseño también se han introducido una serie de registros intermedios, diferentes a los anteriores, que son los que componen el *pipeline*, utilizados para tener una mejor gestión y encaminamiento en el camino de una instrucción. Estos registros se colocan entre cada una de las etapas, y se comportan como compuertas para almacenar la información momentáneamente entre las etapas. Así, tenemos cuatro registros intermedios: IF/ID, ID/EX, EX/MEM y MEM/WB. En la Figura 3 se puede ver un esquema de las diferentes etapas que componen el procesador y sus registros intermedios.

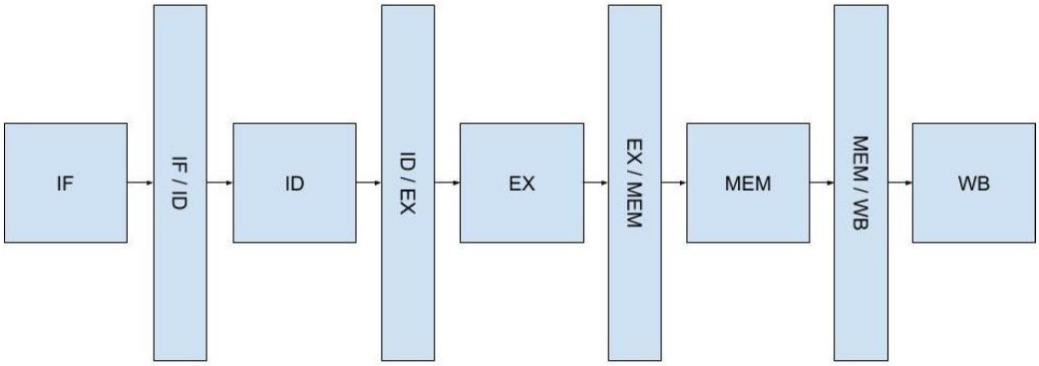


Figura 3: Etapas del procesador con pipeline [3]

Además, el SRG ha implementado un módulo paralelo a las etapas del *pipeline* dedicado a las instrucciones de traza. El camino que sigue este tipo de instrucciones en las dos primeras etapas va a ser exactamente el mismo que para una instrucción nominal. En cambio, como las instrucciones de traza no necesitan ejecutarse, ni acceder a la memoria, ni tampoco escribir en registros, lo que se ha hecho es un camino para extender esta instrucción a través de los registros intermedios de las etapas restantes, llegando hasta un registro final cuando se termina de procesar, como se indica en la siguiente Figura 4.

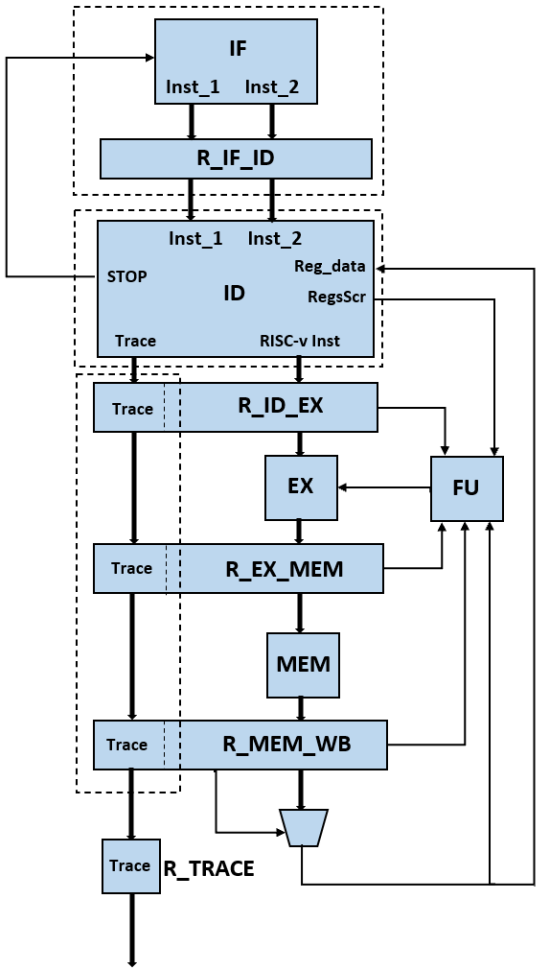


Figura 4: Etapas del procesador con pipeline y con instrucciones de traza [3]

Cabe señalar, que también se han añadido algunas modificaciones para el correcto funcionamiento del procesador teniendo en cuenta el trabajo de la patente. Una de ellas es la de coger las instrucciones de 2 en 2, para así aumentar la eficiencia del procesador, ya que en el caso de que se tenga en el cauce una instrucción de traza, habría un momento en el que la etapa de ejecución no trabajaría, disminuyendo el rendimiento del procesador y malgastando recursos. Otra modificación ha sido la lógica adicional al procesador encargada de poder elegir qué instrucción se debe de ejecutar primero. [2] [3]

Esta es la estructura general del procesador que ha desarrollado el grupo de investigación espacial de la UAH. A continuación, se van a explicar las partes del procesador, pero solo centrándose en el camino donde se ejecutan instrucciones nominales.

2.3.1. FASES DEL PROCESADOR

IF – INSTRUCTION FETCH

Esta etapa se va a encargar de coger dos instrucciones de la memoria de instrucciones e introducirla en el flujo de datos. Para esto se hace uso de un contador de programa que va buscando las instrucciones según se requieran y las introduce en el registro intermedio IF/ID para que estén listas en la siguiente etapa.

ID – INSTRUCTION DECODE

Como se ha mencionado anteriormente, una instrucción posee una serie de campos los cuales nos dan información de qué es lo que hace la instrucción, de qué registros está compuesta y dónde tiene que encaminarse.

Se le ha agregado para esto un módulo llamado ***Forwarding Unit***, que como su propio nombre indica, es una unidad de adelantamiento de datos a etapas posteriores, en el caso de dependencias en instrucciones consecutivas. A la *Forwarding Unit* están conectados todos los registros intermedios para el caso que tenga que adelantarse algún dato.

Además, esta unidad se encarga de la detección de amenazas entre las instrucciones del *pipeline* y del control de las funcionalidades del resto de las etapas posteriores. La Unidad de Control de Amenazas (en inglés, ***Hazards Control Unit* - HCU**), se ha diseñado para detectar varios tipos de amenazas, en la que se incluye un mecanismo de parada en caso de detección de algún error en el procesador. En la Figura 5 se puede observar el esquema presentado por la **HCU**, donde las dos instrucciones (la de traza y la nominal) siguen el mismo camino antes de la **CTU**, ***Control Trace Unit***, (en español, Unidad de Control de Traza). En la CTU existe una máquina de estados que indica lo que debe hacer el procesador. La *Hazard Unit* (HU) se coloca antes de la CTU para que así le dé tiempo al procesador a trazar la amenaza. Después de la CTU, las

instrucciones se encaminan dependiendo de si es de traza o no. Las instrucciones que siguen el camino de no-traza, pasan por la unidad de control (***Control Unit***, **CU**). Esta unidad se encarga del encaminamiento de los trozos decodificados de la instrucción. En caso de parada debido a una amenaza, el multiplexor, que aparece después de la CU, impide que se propague la instrucción, introduciendo la instrucción NOP, es decir una burbuja, evitando así enviar datos no válidos.

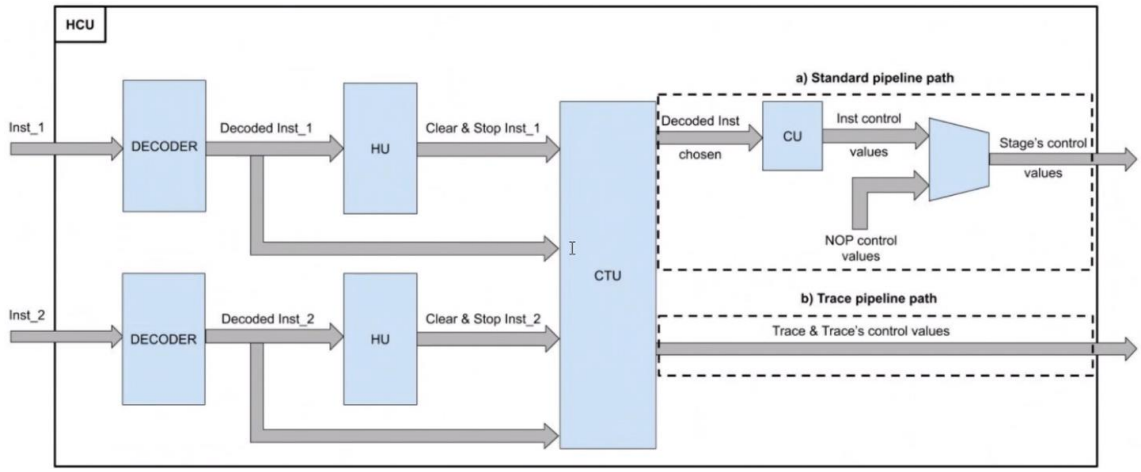


Figura 5: Hazard Control Unit [3]

EX – EXECUTION

En la etapa de ejecución se producen todas las operaciones de cálculo de las instrucciones. Está compuesta por la **ALU** (*Arithmetic Logic Unit*), que es la que se encarga de realizar todas las operaciones de cálculo, y por un módulo **Comparador**, encargado de tomar las decisiones de salto. Esta etapa se explica con más de detalle posteriormente, dado que, como se verá más adelante es la etapa que más tiempo tarda en procesar la información, y, por lo tanto, es un importante caso de estudio para este trabajo.

MEM – MEMORY

En la etapa de memoria, la instrucción plantea la necesidad de acceder a memoria, bien porque tenga que leer algún dato de algún registro o guardar algún dato en la memoria de datos.

WB – WRITE BACK

El final del procesador acaba con esta etapa, que únicamente lo que hace es acceder al banco de registros para escribir en uno de ellos el resultado de una operación. Como es posible que haya otras instrucciones que deseen acceder al banco de registros para leer un registro, se ha programado el sistema para que en una mitad de un ciclo de reloj se pueda acceder al banco de registro para lectura y en la otra mitad del ciclo de reloj, se acceda a éste para la escritura, según recomienda Patterson & Hennessy [5].

Por último, solo indicar, que el diseño del *pipeline* viene acompañado por un módulo que trabaja como interfaz con la memoria, llamado *Tilelink*, y un divisor de frecuencia, aunque no lo tenemos en cuenta en este estudio ya que no es relevante para la mejora del propio *pipeline*, aunque en las pruebas e implementación final, sí que se incluirán. [2] [3]

Pues bien, este es el diseño del procesador con el que partimos para el estudio y mejora en las partes donde sea más lento y menos eficiente. En el siguiente apartado se explicarán las herramientas utilizadas para hallar las partes del sistema a mejorar.

2.4. VIVADO

En el diseño de sistemas electrónicos se utilizan herramientas para realizar los diseños, validaciones, simulaciones e implementaciones de los sistemas para comprobar que sean correctos y en caso de encontrar fallos, poder hallarlos en fases tempranas del proyecto para que no se haga muy tedioso el proceso.

En este trabajo, la herramienta utilizada para este fin es *Vivado Desing Suite*, creada por la compañía Xilinx. Esta herramienta fue desarrollada para la síntesis y análisis de circuitos HDL (*Hardware Description Language*), es decir, lenguajes para la descripción de circuitos electrónicos. El lenguaje que se ha utilizado en este proyecto es VHDL y la implementación se ha realizado en una placa Nexys 4 DDR.

En los **anexos** se explica paso a paso la instalación y configuración de la herramienta Vivado para este proyecto. En este apartado se habla de las herramientas que ofrece Vivado para la mejora del procesador en cuestión, que es el caso de estudio en este proyecto. Se hace hincapié en los reportes de tiempo que brinda Vivado, ya que, al querer mejorar la velocidad del procesador, interesa hallar las etapas en las que el procesador es menos eficiente.

2.4.1. REPORTE DE TIEMPOS

La herramienta Vivado tiene la opción de realizar un reporte de tiempos del sistema diseñado, diseccionando cada componente y cada nodo del circuito. Seleccionando una serie de filtros e interpretando cada resultado que se muestra, se puede averiguar qué partes del circuito podrían mejorarse, o incluso detectar errores que de otro modo no se podría comprobar.

A continuación, se explican una serie de términos que se pueden encontrar en este reporte de tiempo, tanto en su configuración como en los resultados que ofrece. [6] En el ANEXO 4: REPORTE DE TIEMPO DE VIVADOse explica cómo obtener estos reportes y sus diferentes configuraciones. Cabe señalar que en este estudio se ha ido comprobando las modificaciones en la fase de la síntesis, ya que se ahorra tiempo a la hora de ir probando los cambios hechos porque a nivel de cómputo, la ejecución de la síntesis de un circuito es más rápida que la implementación.

La configuración que se ha utilizado para obtener el reporte de tiempo con los datos que interesan es la que podemos observar en el apartado *Timer Settings* (en español, configuración del temporizador) tal y como muestra la siguiente Figura 6:

Figura 6: Reporte de tiempos - Timer Settings

A continuación, se explican brevemente estos términos, aunque en el ANEXO 4: REPORTE DE TIEMPO DE VIVADOse explica con más detalle estos y otros términos que podemos configurar en la herramienta Vivado. Siguiendo los pasos del anexo se obtienen los datos de los tiempos de ***Slack*** para el tiempo de ***Setup*** y de ***Hold***.

- El tiempo de ***SETUP***, o ***RECOVERY***, es la cantidad de tiempo necesaria para que una señal tenga un valor estable **antes** de un cambio de flanco de reloj, para así conseguir el valor lógico esperado en la señal de salida.

- El tiempo de ***HOLD***, o ***REMOVAL***, es la cantidad de tiempo necesaria para que una señal tenga un valor estable **después** de un cambio de flanco de reloj, para así conseguir que el valor lógico esperado en la señal de salida no se altere.

En la siguiente figura 7 se puede ver de manera gráfica dónde se encuentran visualmente los tiempos de *Setup* y *Hold*.

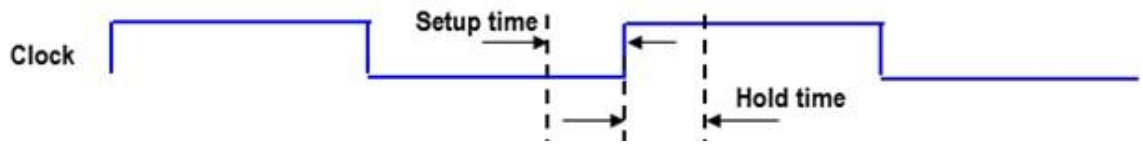


Figura 7: Setup Time y Hold Time

Una vez que se ejecuta el reporte de tiempos, que se muestra en el ANEXO 4: REPORTE DE TIEMPO DE VIVADO, aparecen unos conceptos importantes a la hora de configurar la herramienta para obtener los reportes que se desean. Estos campos son *Path delay type* y *Multi-Corner Configuration*. En las siguientes Figura 8 y Figura 9, aparece la localización de éstos.

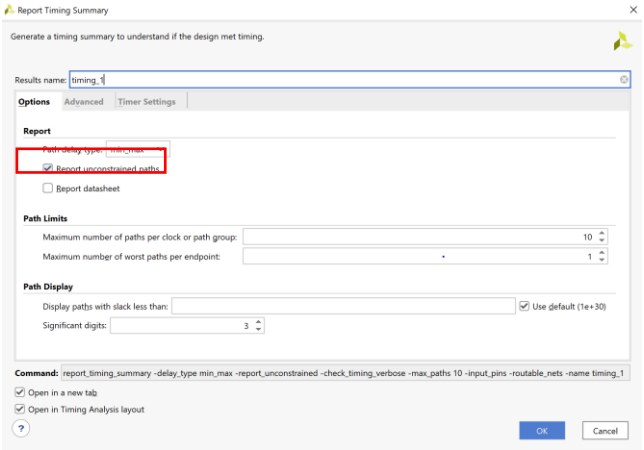


Figura 8: Configuración Report Timing Summary - Options - Path delay type

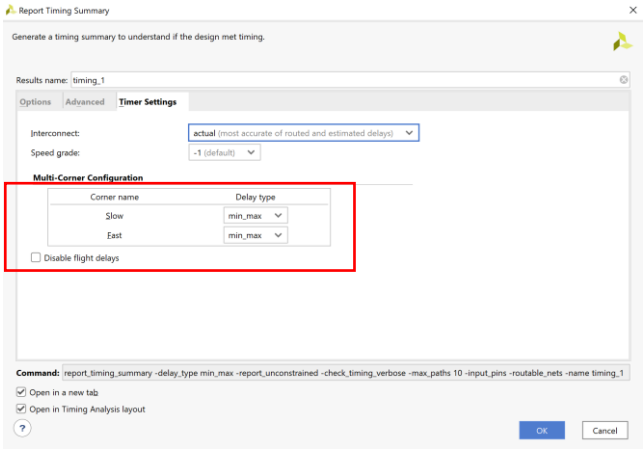


Figura 9: Configuración Report Timing Summary - Timer settings - Multi-Corner Configuration

- ***Path delay type – max***: indica que los valores de retardo máximo son usados para calcular el retardo de la ruta de datos, que corresponde con el análisis de *setup/recovery*.

- ***Path delay type – min***: indica que los valores de retardo mínimo son usados para calcular el retardo de la ruta de datos, correspondiente al análisis de *hold/removal*.

- ***Slow corner***: indica el comportamiento más lento del sistema posible para cada uno de los caminos de la ruta de datos.

- ***Fast corner***: indica el comportamiento más rápido del sistema posible para cada uno de los caminos de la ruta de datos.

Por lo tanto, el peor caso de retardo para comprobar el tiempo de ***Setup*** se obtiene con la configuración de ***Max Delay*** y ***Slow Corner***. Por el contrario, el peor caso de retardo

en el caso del tiempo de ***Hold*** se obtiene al configurar el reporte con ***Min delay*** y ***Fast Corner***.

Una vez que se ha configurado los filtros para obtener el reporte, queda interpretar los resultados que se han obtenido. Para ello se va a explicar los datos obtenidos más relevantes. Para más información, véase el ANEXO 4: REPORTE DE TIEMPO DE VIVADO.

En la pestaña *Timing* se obtienen varios campos que especifican los resultados relativos al reporte de tiempo. Entre estos resultados, tenemos un resumen general de este reporte en el campo llamado ***Design Timing Summary***.

2.4.1.1. *DESIGN TIMING SUMMARY*

Como se puede apreciar en la Figura 10, se han obtenido tres campos diferentes: *Setup*, *Hold* y *Pulse Width* (en español, ancho de pulso). Anteriormente ya se ha definido *Setup* y *Hold*, pero quedaría por definir *Pulse Width*.

- **Pulse Width**, se refiere a cuánto debe ser el ancho del pulso del reloj como mínimo para el correcto funcionamiento del sistema.

Dentro de cada uno de estos tres términos, se obtiene el dato con el que trabajaremos principalmente para averiguar las posibles mejoras del procesador: **WNS – *Worst Negative Slack*** (en español, la peor holgura negativa), que está relacionado directamente con el peor caso de ejecución posible, WCET, del cual se ha hablado anteriormente.

- ***Slack Time***: es la cantidad de tiempo en el que una tarea puede ser retardada sin que afecte al retardo de otra tarea o al sistema completo.

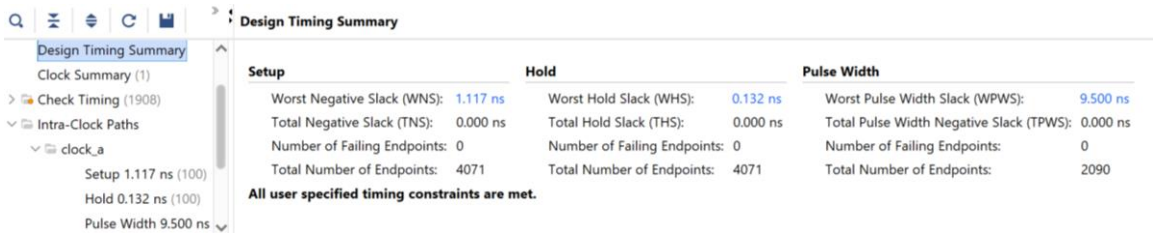


Figura 10: Report Timing - Design Timing Summary

Por lo tanto, con este tiempo obtendremos el peor caso de ejecución posible, **WCET**, tanto para el sistema completo, como para cada uno de los caminos en la ruta de datos. Tanto el WNS (*setup*), como el WHS (*hold*), tienen que ser resultados positivos, ya que un resultado negativo, conllevaría coger un valor de dato erróneo, con lo que la ejecución del sistema no sería válida.

2.4.1.2. *PATHS* CON WCET

Otro campo que nos lleva a inspeccionar cada ruta que toma un dato es ***Intra-Clock Paths***, donde se encuentra el reloj (o relojes) usados en el sistema. En este caso solo se ha definido un reloj para el sistema. A su vez, dentro del reloj se encuentran los campos *setup*, *hold* y *pulse width*, donde se localizan las diferentes rutas que sigue un dato, en las que se ha sacado el reporte de tiempo para el peor caso de estudio.

En este apartado se pueden hallar las rutas más lentas del procesado, pudiendo visualizar su esquemático, la cantidad de componentes por la que pasa (niveles lógicos), el camino que sigue desde el origen al destino, la cantidad de retardo que tiene el camino a nivel lógico y a nivel de nodos, entre otros datos que se pueden apreciar en la siguiente Figura 11. En el ANEXO 4: REPORTE DE TIEMPO DE VIVADO se explica detalladamente cada dato.

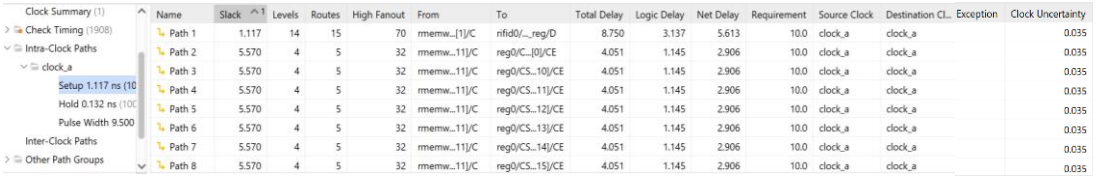


Figura 11: Report Timing - Intra-Clock Paths - Clock Setup

Cuando se ha localizado un camino que se quiere estudiar, se puede visualizar en detalle la ruta y los tiempos que sigue en cada etapa de su ruta, haciendo doble clic sobre el camino escogido. En la siguiente Figura 12 se visualizan los detalles de la ruta seleccionada.

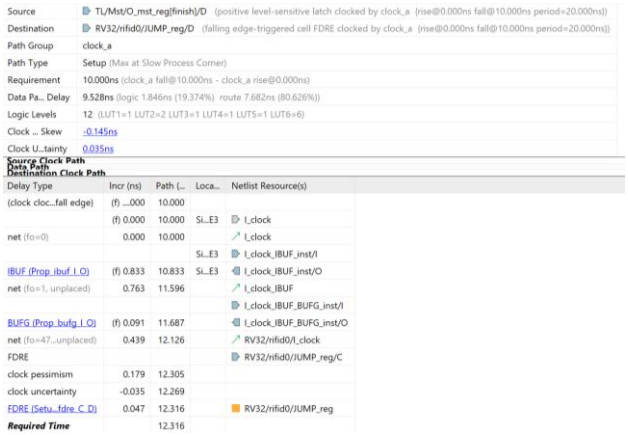


Figura 12: Report Timing - Path

Hasta aquí llega la base teórica de este proyecto, donde se han explicado los antecedentes del proyecto partiendo de la patente por la cual se desarrolla, pasando por la definición de la arquitectura del procesador en RISC-V, la explicación básica del procesador en cuestión y un resumen de la herramienta utilizada para el análisis y desarrollo del procesador, Vivado, donde se han detallado las partes más relevantes del programa.

3. DESCRIPCIÓN EXPERIMENTAL

En este apartado, se explica el desarrollo experimental del trabajo. Una vez que ya se han estudiado todos los conceptos relativos a este proyecto, se ha alcanzado un nivel adecuado de la herramienta Vivado y se entiende el funcionamiento del procesador, se ha procedido a realizar las modificaciones adecuadas para la mejora del procesador. Se realizan varios análisis donde se localiza la parte del procesador con las rutas más lentas del sistema, para así conseguir realizar mejoras que pueda aumentar la frecuencia de funcionamiento del sistema.

3.1. PRIMER ANÁLISIS

En un primer contacto con el análisis que ofrece Vivado, los pasos a seguir, explicados en detalle en el ANEXO 3: MANUAL DE USUARIO DE VIVADO son los siguientes:

- Crear el proyecto
- Cargar los ficheros
- Crear el fichero de restricciones, donde establecemos un reloj "clk = 20 ns"
- Ejecutar la síntesis
- Ejecutar el reporte de tiempos (*Report Timing Summary*). La configuración establecida en el reporte de tiempos será la siguiente (ver Figura 13, Figura 14 y Figura 15):

    - ***Path delay type: min_max***
    - ***Maximum number of paths per clock or path group*: 100,** para así poder comprobar más rutas y realizar un análisis más eficiente, ya que por defecto aparece 10.
    - ***Interconect: actual***
    - ***Multi-corner configuration*: *min_max* en ambos *slow* y *fast corners***

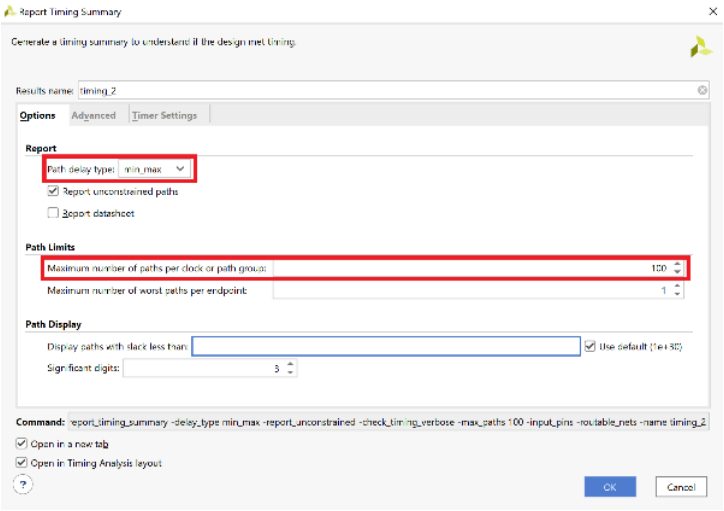


Figura 13: Configuración Report Timing – Option

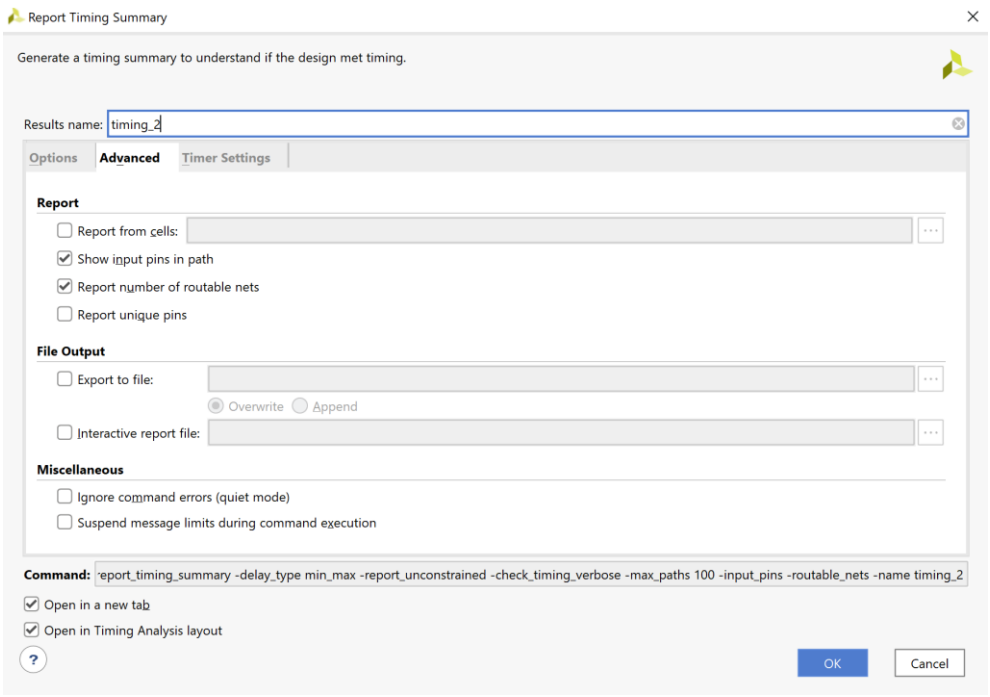


Figura 14: Configuración Report Timing – Advanced

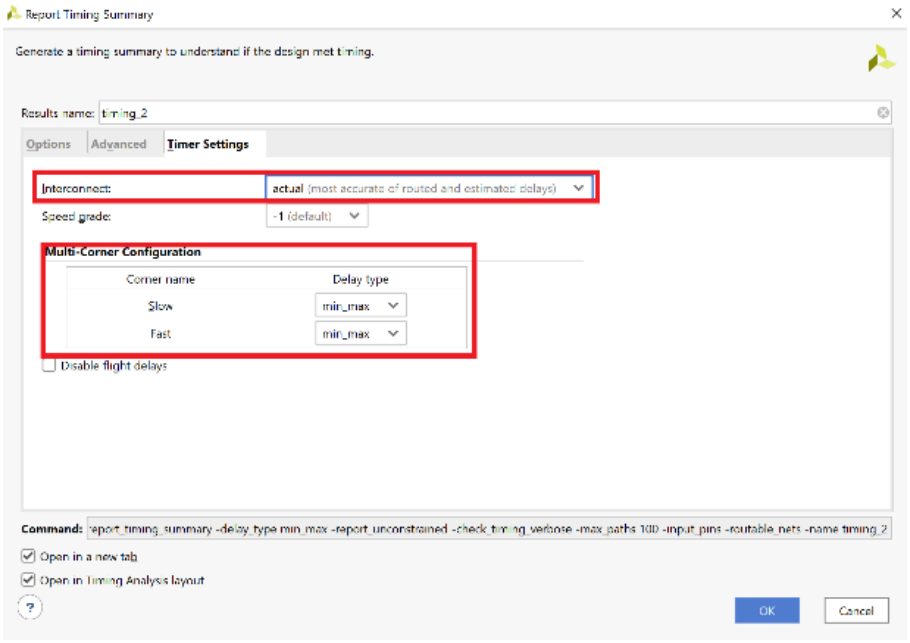


Figura 15: Configuración Report Timing - Timer settings

El reporte que se obtiene con esto es el mostrado en la siguiente figura. Como se puede apreciar se tiene un tiempo de *Slack* **negativo**, tanto en *Setup* como en *Hold*, por lo que las restricciones de tiempo no se cumplen (*Timing constraints are not met*). Esto significa que pueden darse problemas en el sistema y generar resultados no validos a la salida. En la siguiente Figura 16 se ve en detalle el reporte analizado.

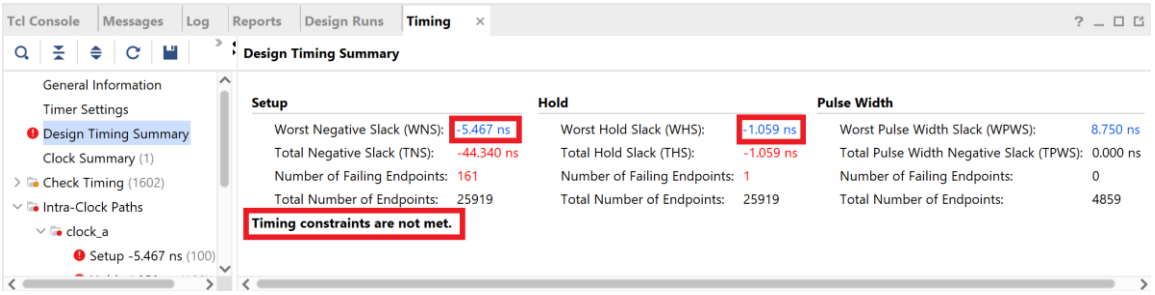


Figura 16: Report Timing - Design Timing Summary - WNS (No se cumplen las restricciones de tiempo)

Para comprobar en qué rutas se están dando estos tiempos de *Slack* negativos, simplemente hay que hacer clic en el número indicado por WNS (resaltado en azul), para comprobar qué rutas se refieren al tiempo de *Setup*, o, para el caso del *Hold*, en WHS.

Al hacer esto, se redirige a la sección donde se presentan las rutas con peores tiempos de ejecución. En este caso la sección se encuentra en *Intra-Clock Paths → clock_a → Setup.* Esta es una manera en la que se pueden comprobar los detalles de las rutas que no cumplen las restricciones de tiempo. Se puede apreciar en la Figura 17, que todas las rutas que no cumplen estas restricciones tienen como origen el módulo *Tilelink*, TL.

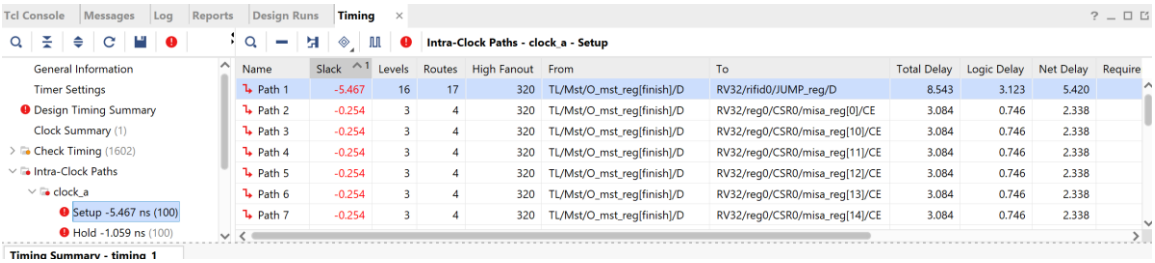


Figura 17: Report Timing - Setup WNS Paths

Quedado que este trabajo se centra en la mejora del *pipeline*, y como el resultado tiene presente una parte que no corresponde al *pipeline*, se ha prescindido del módulo TL para realizar el análisis y las mejoras, y así tener una visión más específica de lo que se está intentando mejorar.

Para ello, se ha realizado la síntesis y el análisis solo de la parte que se ha llamado como *RV32: PIPELINE*, es decir, se establecerá como *Top Level* el *pipeline* y así realizar los análisis sobre este componente. Para colocar el *pipeline* como *Top Level* hay que hacer clic derecho sobre el fichero *RV32: pipeline*, en el apartado *Sources* y hacer clic en *Set as Top*. Aparecerá un símbolo a la izquierda del nombre con 3 cuadrados, indicando que está como *Top Level*. Una vez hecho esto, hay que ejecutar la síntesis de nuevo, tal y como se indica en la siguiente Figura 18.

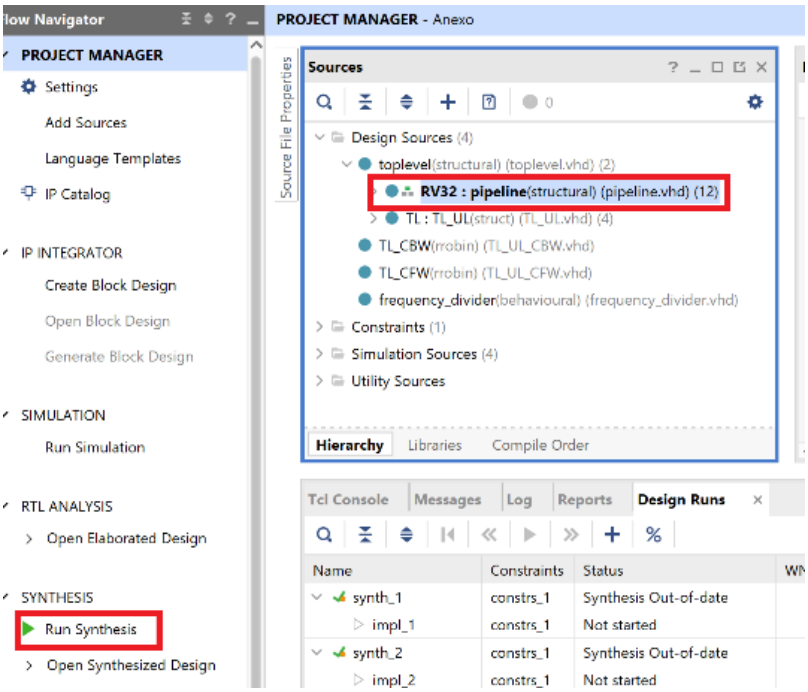


Figura 18: Síntesis RV32: pipeline

3.2. SEGUNDO ANÁLISIS

Cuando se ha ejecutado la síntesis del circuito de nuevo, se realiza un nuevo reporte de tiempos. En la siguiente Figura 19 se observa que esta vez el mensaje mostrado es que las restricciones de tiempo se han cumplido, pero el *Slack* obtenido en el *Setup* es de **0.00 ns**.

Para analizar mejor este resultado, se ha comprobado la ruta completa de uno de esos caminos que resultan **0.00 ns**. De nuevo, haciendo clic en el resultado (resaltado en azul), se dirige al apartado donde aparece las diferentes rutas analizadas, y haciendo doble clic sobre la primera de ellas, aparece en la sección *Workspace* de Vivado, el detalle de tiempo de esa ruta específica (ver Figura 20).

Esta vez, aparece un nuevo tiempo en el reporte, *Time Borrowing*. Esto ocurre en las rutas en las que se accede a la CU, *Control Unit*, ya que esta unidad se había configurado para que funcionara a nivel, por lo que el peor caso posible es cuando llega el dato, por lo tanto, existe este *time borrowing*, que lo que hace es esperar a que el dato esté listo para continuar.

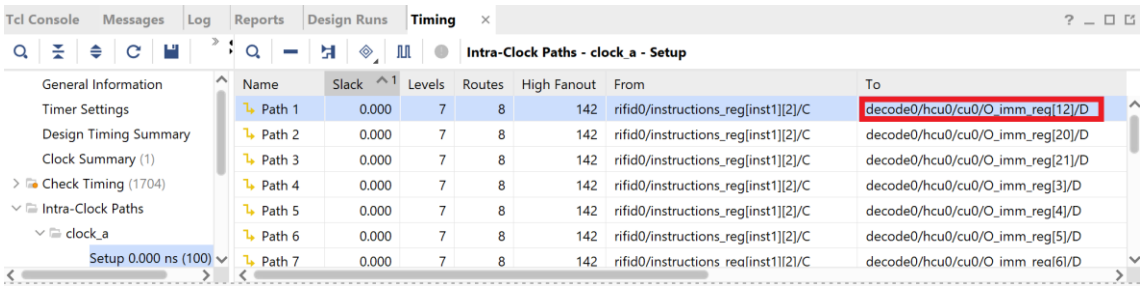


Figura 19: Path timing - Path 1 route

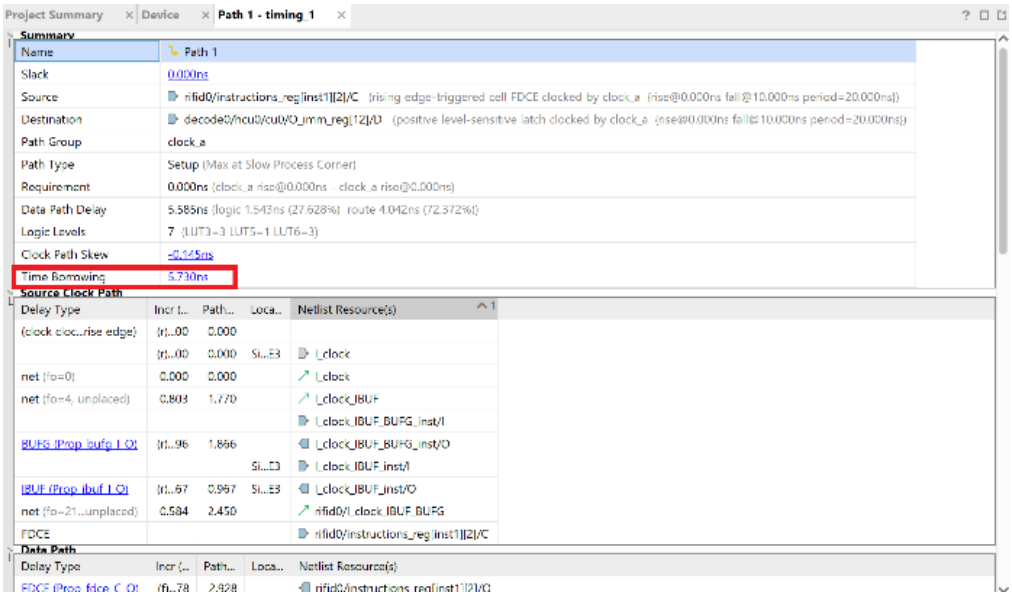


Figura 20: Path timing - Time Borrowing

Para solventar este incidente de espera, se ha quitado la secuencialidad de este componente para que funcione sin tener que esperar al dato, es decir se ha eliminado el reloj en este componente, ya que no es necesario para la sincronización del sistema.

Esto se ha realizado accediendo al código de la CU, que está dentro de la HCU, *Hazard Control Unit*, que a su vez está dentro de la etapa de decodificación. En el código se podía ver que se ejecutaban las instrucciones dependiendo del nivel alto del reloj. Quitando este reloj, ahora las instrucciones se leerán independientemente del reloj, ejecutándose únicamente aquellas que cumplen los requisitos impuestos en el código.

Después de estos cambios y guardar el proyecto, se vuelve a realizar la síntesis y el reporte para comprobar los nuevos resultados. Como indica la siguiente Figura 21 procedente del reporte de tiempo se tiene un **WNS de 0.875 ns**.

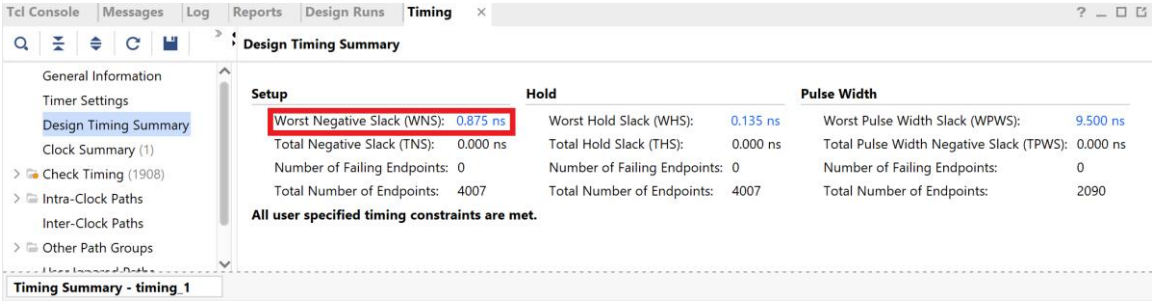


Figura 21: Report Timing - Design Timing Summary - WNS - Se cumplen las restricciones de tiempo

Hasta aquí se han resuelto algunos problemas que se tenían ya que los resultados obtenidos no eran correctos o coherentes. A partir de este momento lo que se va a seguir haciendo es analizar en detalle los resultados obtenidos para hallar así un modo de aumentar la eficiencia del *pipeline*.

3.3. TERCER ANÁLISIS

En el próximo análisis lo que se ha tratado es de buscar diferentes formas de aumentar la eficiencia del sistema. En resumen, se trata de buscar que el peor caso de ejecución posible WCET, sea maximizado, es decir, lo que se busca es que el tiempo WNS que indica el *Setup*, se lo mayor posible.

Para ello, hay que dirigirse a las rutas que tienen el peor caso de ejecución, haciendo clic en el tiempo WNS (resaltado en azul). Esto redirige a la ventana que se aprecia en la siguiente Figura 22 donde se pueden analizar varios factores, que indicarán los pasos a seguir. En la imagen aparece el resultado del reporte en cuanto a las rutas que tienen el *slack* más bajo. Se puede apreciar que la primera ruta, es diferente a todas las demás, con una notable diferencia tanto en el tiempo de *slack*, como en los niveles de lógica, la ruta que siguen o el tiempo de retardo a nivel lógico y a nivel de conexión.

Esto da una visión clara de qué partes del procesador son más lentas y donde podría existir una posible mejora. Hay varias formas de reducir el tiempo de ejecución de una ruta. Se podría reducir el nivel de lógica que atraviesa el dato, o reduciendo el conexionado de la ruta, o simplificando un componente, etc.

Siguiendo con la siguiente Figura 22, y fijándose en el primer camino, se puede apreciar que es en éste donde se debe realizar la modificación. La ruta que se analiza tiene el origen en el registro intermedio entre la etapa de memoria y escritura. Después se encamina hacia la *Forwarding Unit*, atravesando el *pipeline_logic*. A continuación, se dirige a la etapa de ejecución, haciendo uso del comparador, para volver de nuevo al *pipeline_logic* y finalizar en el registro intermedio entre la etapa de búsqueda y la de decodificación. Este camino se muestra en la Figura 24.

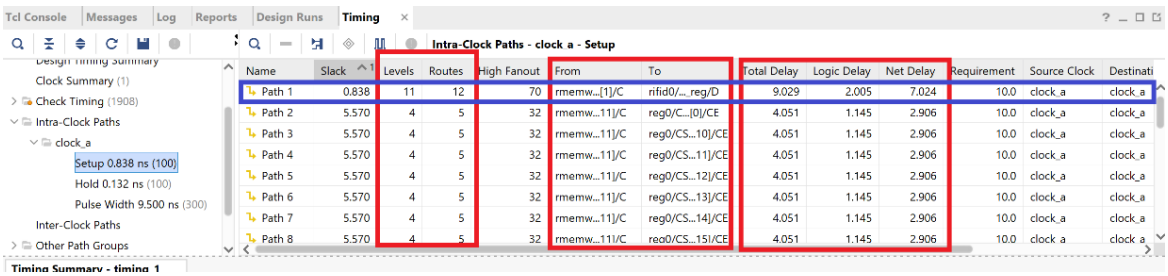


Figura 22: Report Timing - Setup timing WNS - Detalles de la ruta

Este camino es típico de instrucciones de *Branch*, ya que se puede observar que en este camino se calcula la dirección de la siguiente instrucción y se hace uso del comparador para comparar si se tiene que realizar el salto o no.

Se podría en este caso optar por reducir el camino del cableado entre los componentes, que claramente se puede apreciar que es mucho mayor (**en total 7.024 ns**) que el tiempo que tarda la lógica de los componentes (**en total 2.005 ns**). Para ello, se ha

estudiado el funcionamiento en detalle de este tipo de instrucciones, haciendo más hincapié en la parte del comparador, que es donde parece que más se ralentiza la ejecución.

A continuación, se realizará una explicación de las instrucciones de *branch*, y la posterior modificación del comparador una vez entendido el funcionamiento.

3.3.1. INSTRUCCIONES DE BRANCH

Una instrucción de *Branch* es una instrucción que se ejecuta comparando dos registros. Si la comparación es cierta, el contador de programa saltará a la instrucción indicada por una dirección, en caso contrario, si la comparación no es cierta, el contador de programa ejecutará la siguiente instrucción de manera secuencial.

Hay diferentes tipos de comparaciones indicados en la siguiente Figura 23:

Branches	Branch =	B	BEQ	rs1,rs2,imm
	Branch ≠	B	BNE	rs1,rs2,imm
	Branch <	B	BLT	rs1,rs2,imm
	Branch ≥	B	BGE	rs1,rs2,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm

Figura 23: Tipos de Saltos condicionales (Branches) [4]

- **BEQ**: compara dos registros y, si son iguales, salta a la instrucción indicada por el campo “imm” (inmediato)

- **BNE**: compara dos registros y, si son distintos, salta a la instrucción indicada por el campo “imm” (inmediato)

- **BLT**: compara dos registros y, si el primero es menor que el segundo, salta a la instrucción indicada por el campo “imm” (inmediato)

- **BGE**: compara dos registros y, si el primero es mayor o igual que el segundo, salta a la instrucción indicada por el campo “imm” (inmediato)

- **BLTU**: compara dos registros sin signo y, si el primero es menor que el segundo, salta a la instrucción indicada por el campo “imm” (inmediato)

- **BGEU**: compara dos registros y, si el primero es mayor o igual que el segundo, salta a la instrucción indicada por el campo “imm” (inmediato).

Para realizar estas comparaciones, se hace uso de un circuito comparador. Las entradas se componen de dos registros y la salida es una señal que dictamina si se realiza un salto a la dirección de memoria indicada por un inmediato o no. La mejora que se ha realizado radica en el comparador del circuito, ya que se ha comprobado que es uno de los componentes donde el sistema es más lento.

Se ha incluido en el ANEXO 5: CÓDIGO el código del comparador mejorado. Para el código se ha creado una función que se utiliza para convertir un resultado a std_logic, simplificando los operandos que utiliza el comparador para la comparación. El funcionamiento del comparador diseñado es el siguiente:

- Como entradas al circuito se tiene “func”, “op1” y “op2”, y como salida “result”.
    - **func** (entrada de 3 bits): representa la comparación que se hace, si es igual o no, mayor o menor, etc.
    - **op1 y op2** (entradas de 32 bits): son los operandos que se van a comparar
    - **result** (salida de 1 bit): indica si se toma el salto a la instrucción (´1´), o no (´0´), ejecutándose la siguiente instrucción inmediata.

- El proceso que sigue es el siguiente: según la entrada “func”, se ejecuta una de las comparaciones y el resultado será ´1´ cuando se cumpla la condición entre los paréntesis de los operandos, y ´0´, en caso contrario.

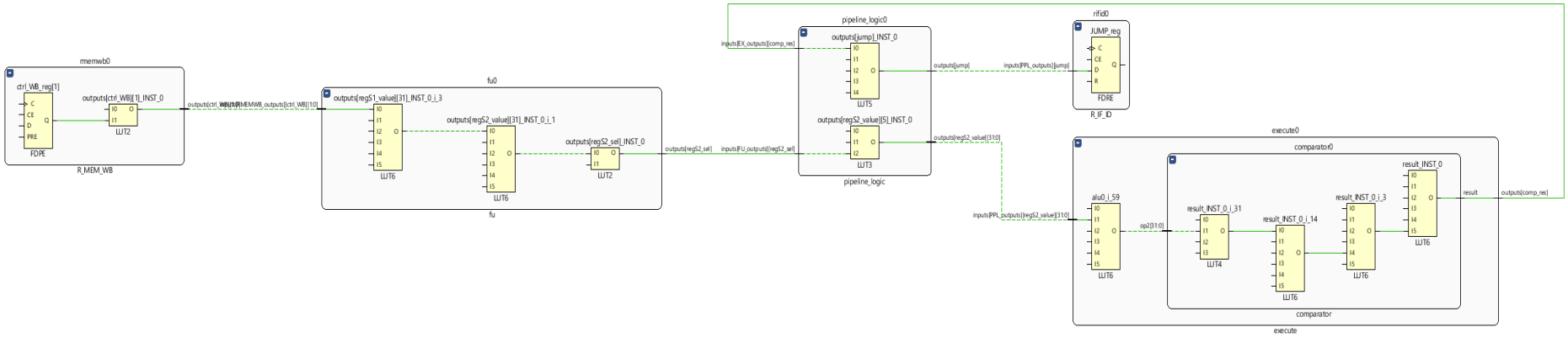


Figura 24: Ruta con el peor WNS

3.4. CUARTO ANÁLISIS

En este último análisis se han comprobado que los resultados obtenidos tras la mejora del comparador han mejorado. Se ha ejecutado de nuevo el reporte de análisis de tiempo y los resultados obtenidos han sido los siguientes (ver Figura 25):

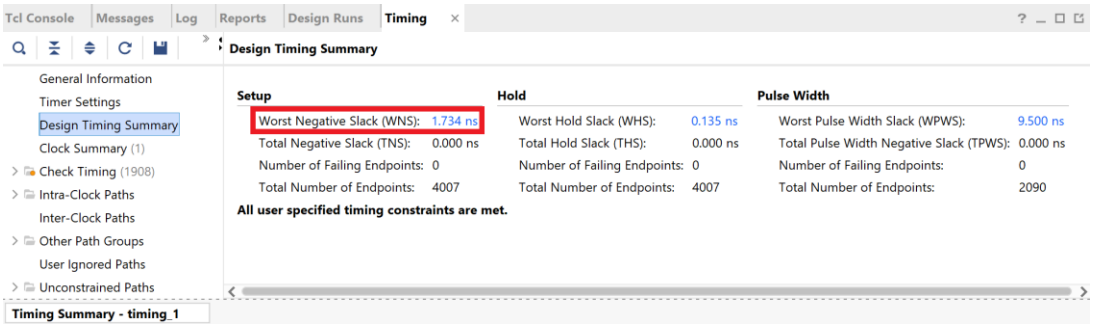


Figura 25: Report Timing - Setup WNS (con la mejora)

Como se observa en la Figura 26, el tiempo WNS es de **1.734 ns**, que comparado con el tiempo que se ha obtenido antes de la modificación del comparador (**WNS = 0.838 ns**), supone una mejora de algo más del doble.

Fijándose en las rutas, se observa que se siguen manteniendo las mismas que anteriormente, pero todas ellas con un mejor tiempo de *slack*, con lo que permitiría aumentar la frecuencia del procesador, para que funcione más rápido sin ningún problema.

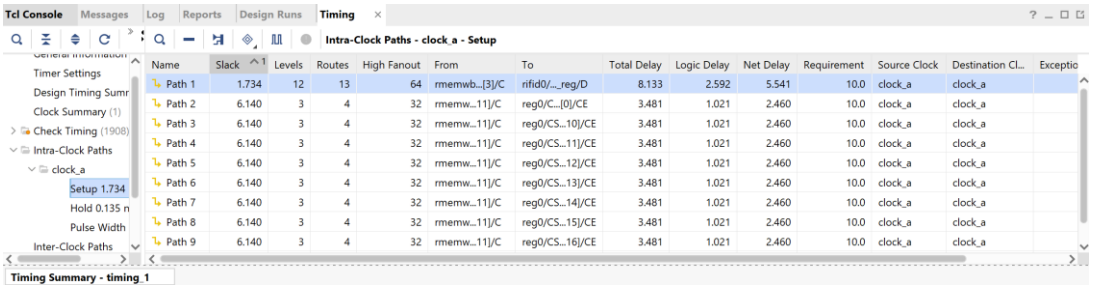


Figura 26: Report Timing - Setup WNS - Paths

Observando la ruta que peor *slack* tiene, si se comparan los tiempos antes y después de la mejora, se observa que el retardo producido por la lógica (*Logic Delay*) ha aumentado, de **2.005 ns a 2.592 ns**, ya que ha aumentado los componentes lógicos. Sin embargo, el retardo debido al conexionado entre componentes (*Net Delay*), ha disminuido bastante, de **7.024 ns a 5.541 ns**. En la Figura 27 se puede ver la ruta en cuestión.

Ya se ha conseguido mejorar el procesador, pero aún está por comprobar su correcto funcionamiento e implementación en la placa, que es de lo que se hablará más adelante.

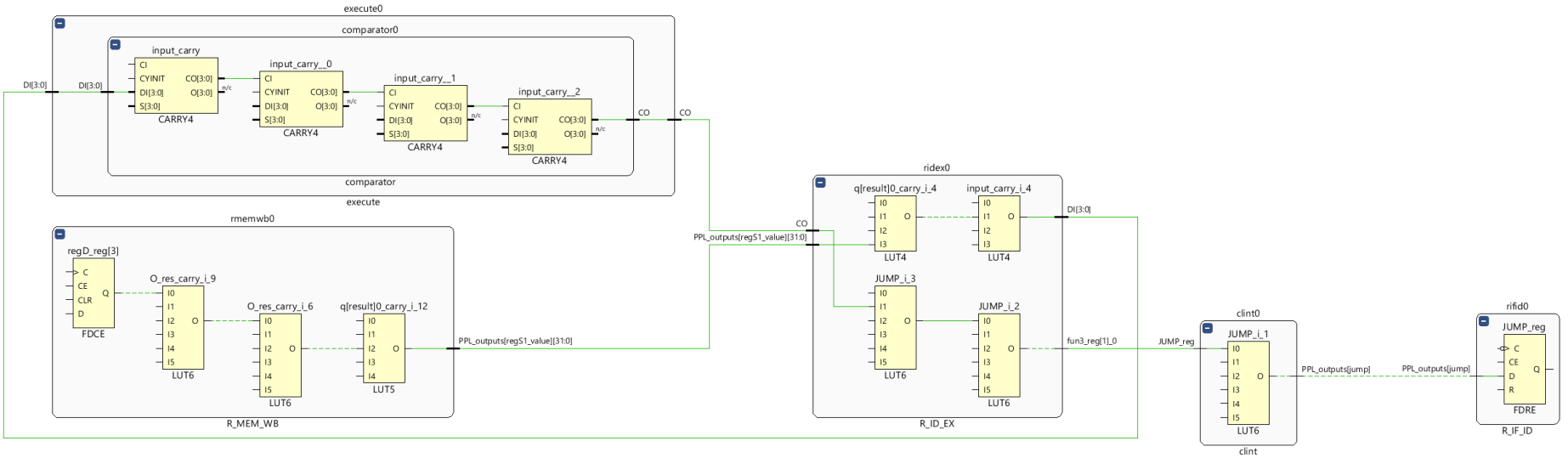


Figura 27: Ruta con el peor WNS mejorada

3.5. REALIZACIÓN DE LAS PRUEBAS

En este apartado se realizan las pruebas de simulación a partir de un *testbench* creado para comprobar que los cambios obtenidos permiten el funcionamiento correcto del sistema. Para la simulación del sistema, se tiene que cambiar de nuevo el *Top Level*, para que ejecute el sistema completo, esto es, añadiendo de nuevo el *Tilelink*. Por lo tanto, en la sección *Sources*, hay que hacer clic derecho sobre *Top Level* y después *Set as Top*, tal y como se muestra en la Figura 28.

Los ficheros utilizados para el *testbench* están localizados en la librería "*work*", donde están guardadas todas las funciones del sistema. El nombre de este fichero es "*Inst_mem4k_inint.vhd*", donde residen los diferentes programas con las instrucciones creadas para realizar las pruebas del procesador. Después de añadirlos, hay que asegurarse que las instrucciones seleccionadas para la ejecución del programa sean las que se quieren comparar. En este caso se ha elegido las instrucciones del programa "*INST_INIT_TB_QUICKSORT8_TRAZA*", la cual debe estar activada para la simulación. Una vez se tiene todo listo, se ejecuta la simulación. Para ello se necesita hacer clic en *Run Simulation*, desde *Flow Navigator* y *Run Post-Synthesis Timing Simulation*.

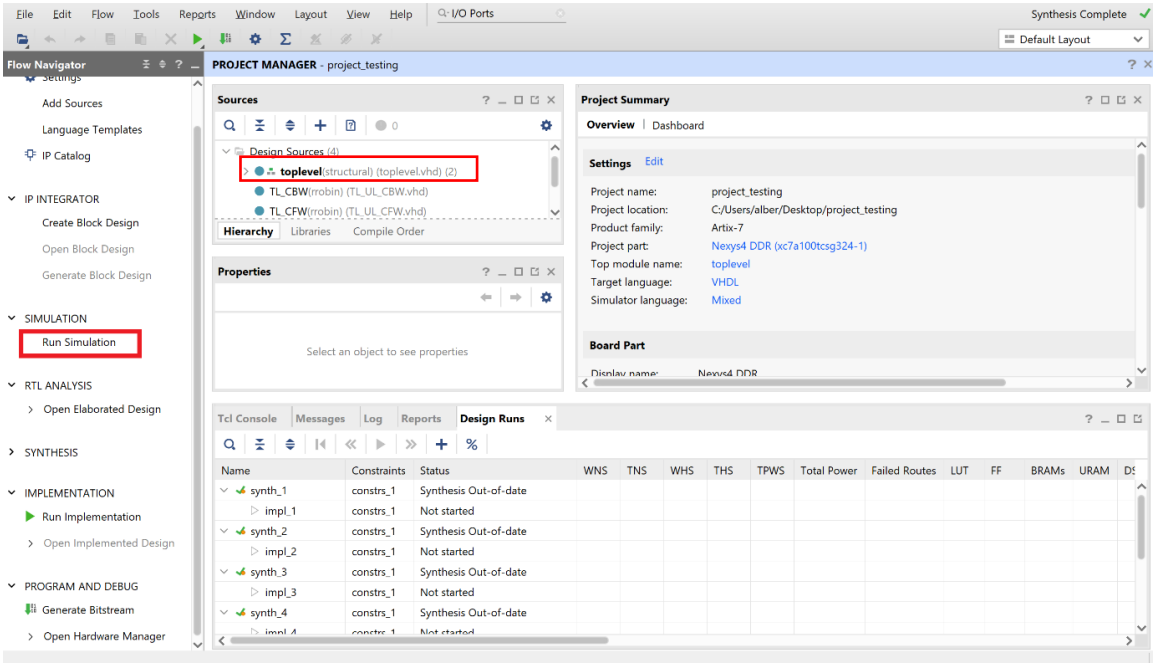


Figura 28: Simulación del sistema mejorado

3.5.1. QUICKSORT8_traza

Este programa creado para realizar las pruebas del procesador RISC-V, se encarga de ordenar de menor a mayor una serie de números. En el ANEXO 5: CÓDIGO se muestra el código en lenguaje ensamblador de este programa. En este programa se introducen también instrucciones de traza para comprobar que el desarrollo de la unidad en paralelo creada para procesar este tipo de instrucciones junto con las instrucciones nominales, referente a la patente y realizado por el grupo de investigación SRG de la UAH, funciona correctamente.

Los números que se introducen en este programa para comprobar que el procesador funciona correctamente son los indicados en los siguientes *arrays*:

a[0] = 39; a[1] = 8; a[2] = 81; a[3] = 89; a[4] = 56; a[5] = 16; a[6] = 61; a[7] = 87;

Por lo tanto, cuando se ejecute este programa en el procesador, se cargarán estos datos en memoria, para luego, mediante comprobaciones, ordenarlos de menor a mayor. Cabe destacar, que este sistema utiliza recursividad, es decir, existe una función que se llama así misma para realizar las comparaciones de si un dato es menor o mayor que otro dato con el que se está comparando.

Una vez se conoce el funcionamiento del programa utilizado para las pruebas, se procede a interpretar los resultados que genera la simulación. Lo primero que hay que tener en cuenta, es que el programa empieza con el ´PC´ (contador de programa) en la posición ´0´. Por lo tanto, es necesario ejecutar la simulación durante tanto tiempo como tarde el programa en ejecutarse, para así intentar buscar el siguiente ´0´ y comprobar que el programa ha acabado correctamente. En la siguiente Figura 29: Ejecución de la simulación se indica el botón de ejecución, en el cual se ha establecido un valor. En este caso, el programa tarda en ejecutarse algo menos de 60 microsegundos, por lo que se ejecutará hasta que el tiempo de simulación, indicado abajo a la derecha de la figura, sea de 60 microsegundos.

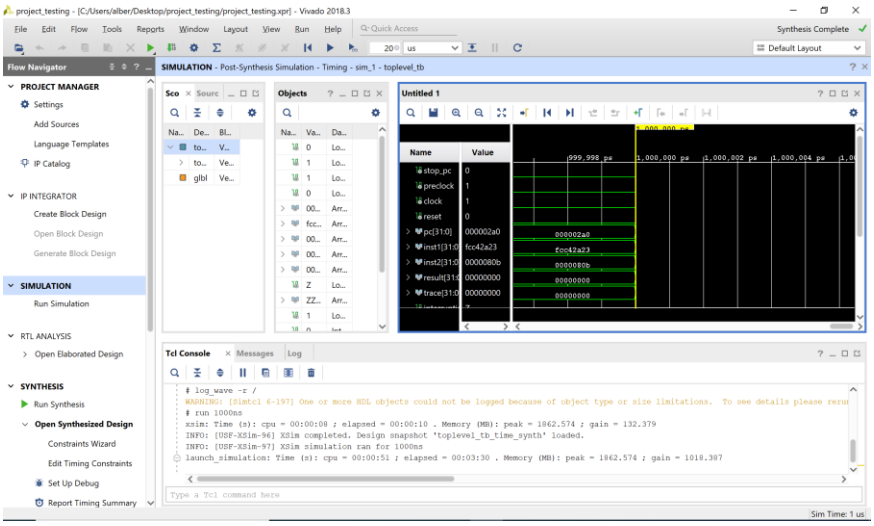


Figura 29: Ejecución de la simulación

Una vez que la simulación ha concluido, se procede a buscar en el contador de programa "PC" el siguiente ´0´. Para ello, en la siguiente Figura 30 se indica como buscarlo. En primer lugar, para tener una vista completa de los resultados, se ha de pulsar el botón . Después de esto se pulsa el icono de la lupa para buscar este valor y se selecciona el dato que queremos buscar, en este caso el "PC", poniendo el campo *Radix* en hexadecimal, para que el valor se muestre en esta base numérica. Después de esto, se hace clic en el botón *Previous*, ya que el cursor que ofrece la simulación se encuentra al final de ésta. Haciendo clic de nuevo a *previous*, el cursor se dirige al principio del programa. Por último, se tiene que definir la salida "*result"* como *signed decimal*, ya que así se podrá interpretar el resultado fácilmente, el cual tendrá que coincidir con los números anteriormente mencionados.

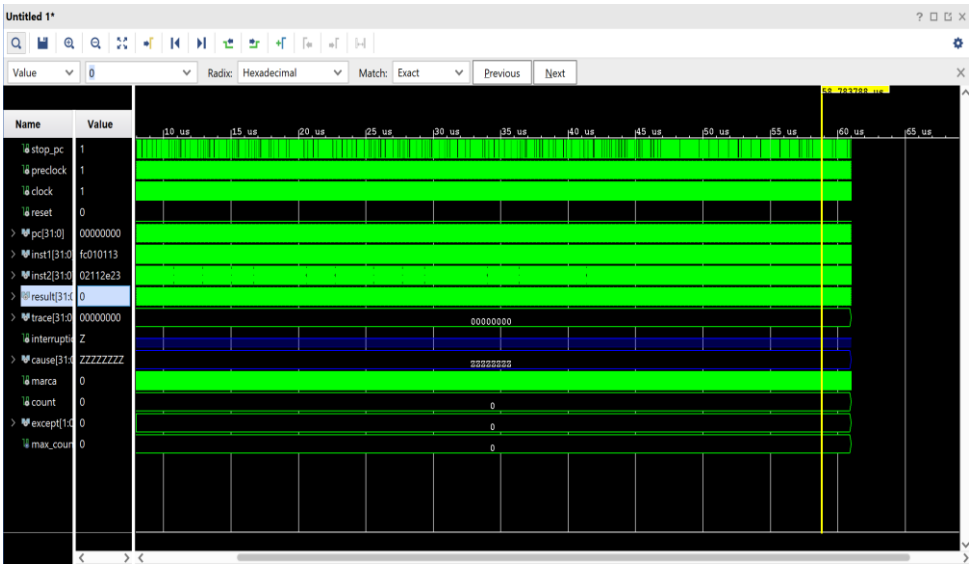


Figura 30: Configuración de los resultados de la simulación

En la siguiente Figura 31, se ha completado la configuración que se ha explicado anteriormente y se ha buscado el final del programa estableciendo el "PC" a ´0´. Además, se ha realizado un *zoom in* para visualizar los resultados de manera sencilla.

Figura 31: Simulaión - PC = 0

Para las comprobaciones, se va a buscar la instrucción referente al “PC = e0”, que es la que va a mostrar el resultado final de los datos buscados. Aunque para una mejor visualización, se buscará “PC = e4”, ya que el dato se leerá correctamente al final de “e0”. A continuación, se muestran las siguientes figuras que se han tomado empezando por el final del programa, mostrando los resultados obtenidos:

- **RESULT = 89**

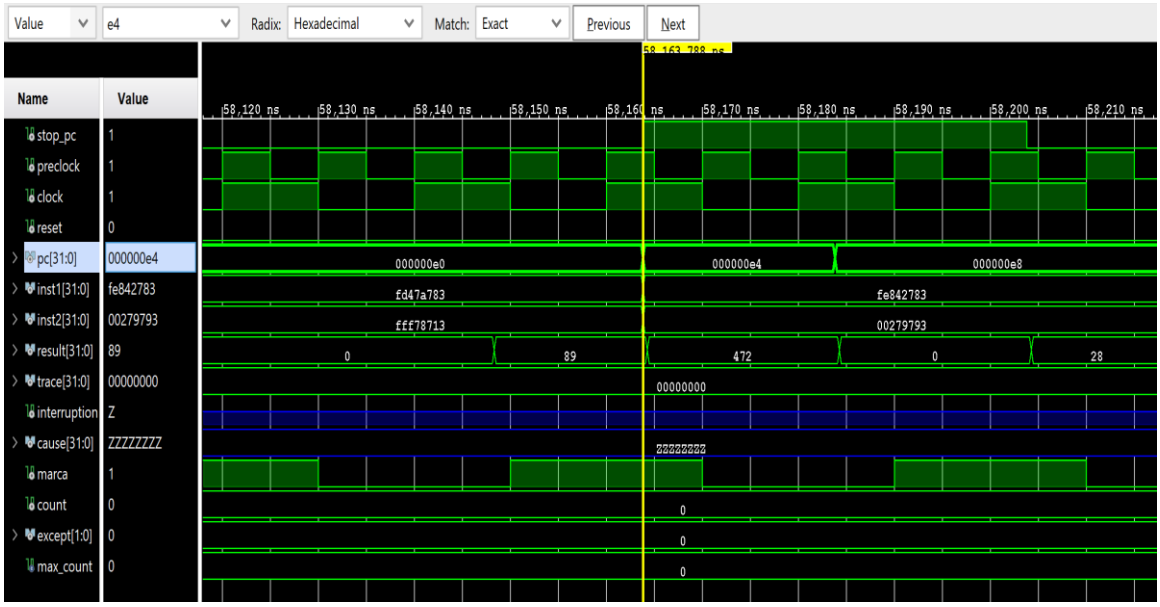


Figura 32: Simulación - PC = e4 - Result = 89

- **RESULT = 87**

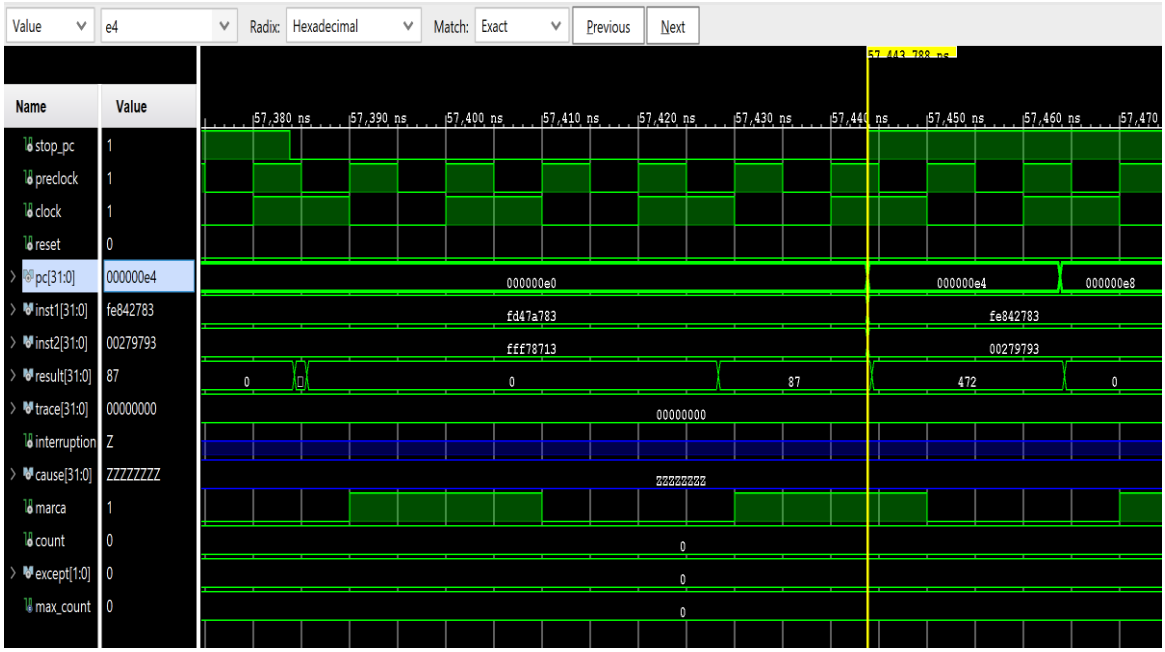


Figura 33: Simulación - PC = e4 - Result = 87

- **RESULT = 81**

Figura 34: Simulación - PC = e4 - Result = 81

- **RESULT = 61**

Figura 35: Simulación - PC = e4 - Result = 61

- **RESULT = 56**

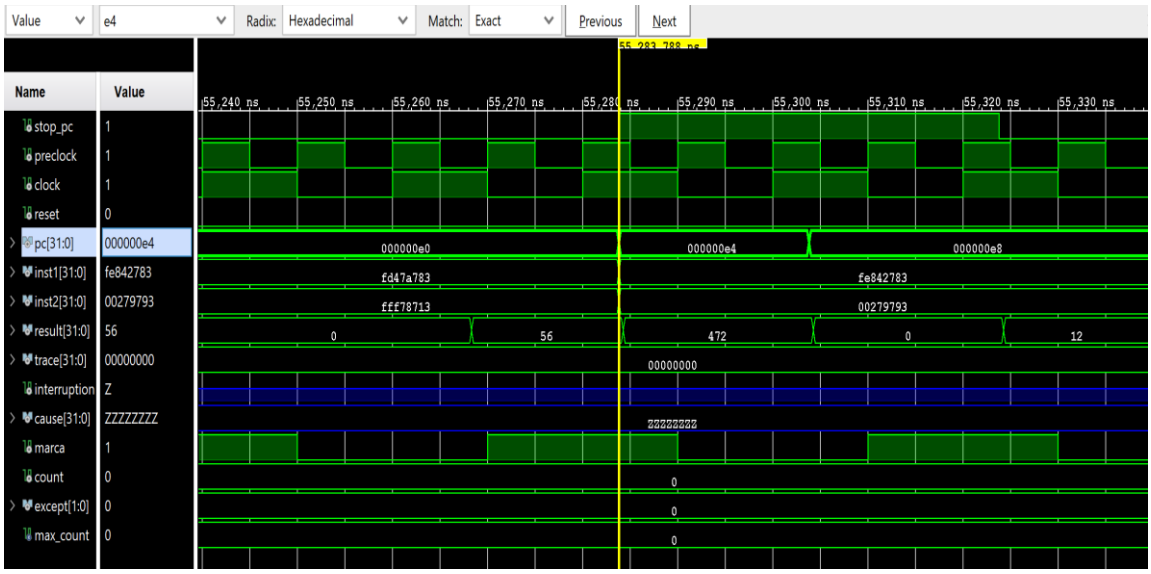


Figura 36: Simulación - PC = e4 - Result = 56

- **RESULT = 39**

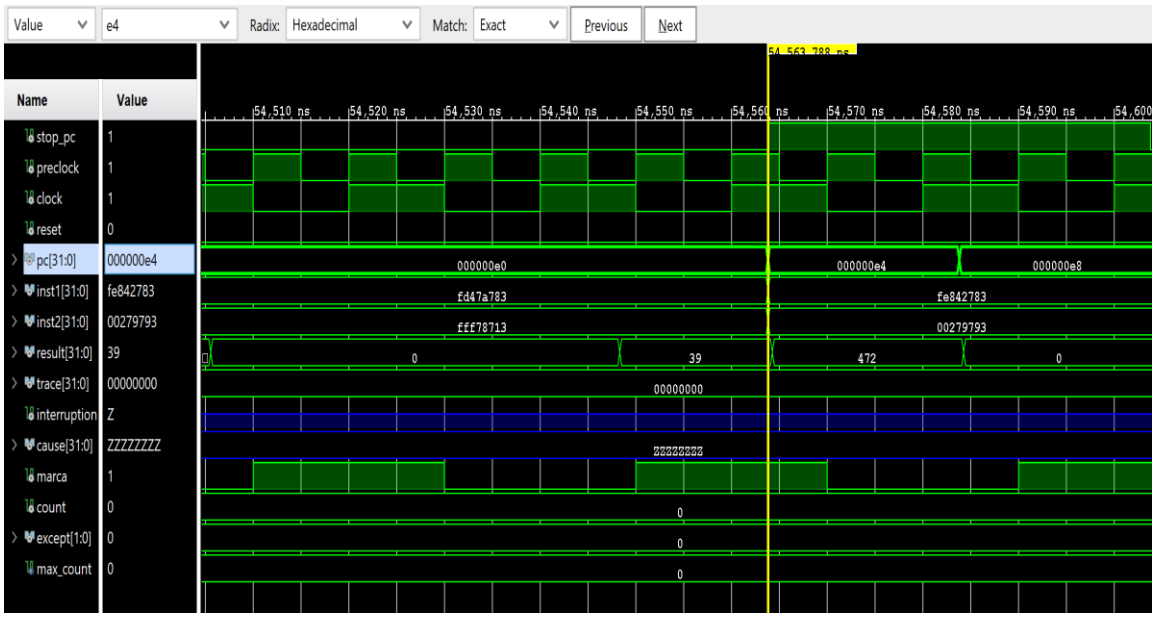


Figura 37: Simulación - PC = e4 - Result = 39

- **RESULT = 16**

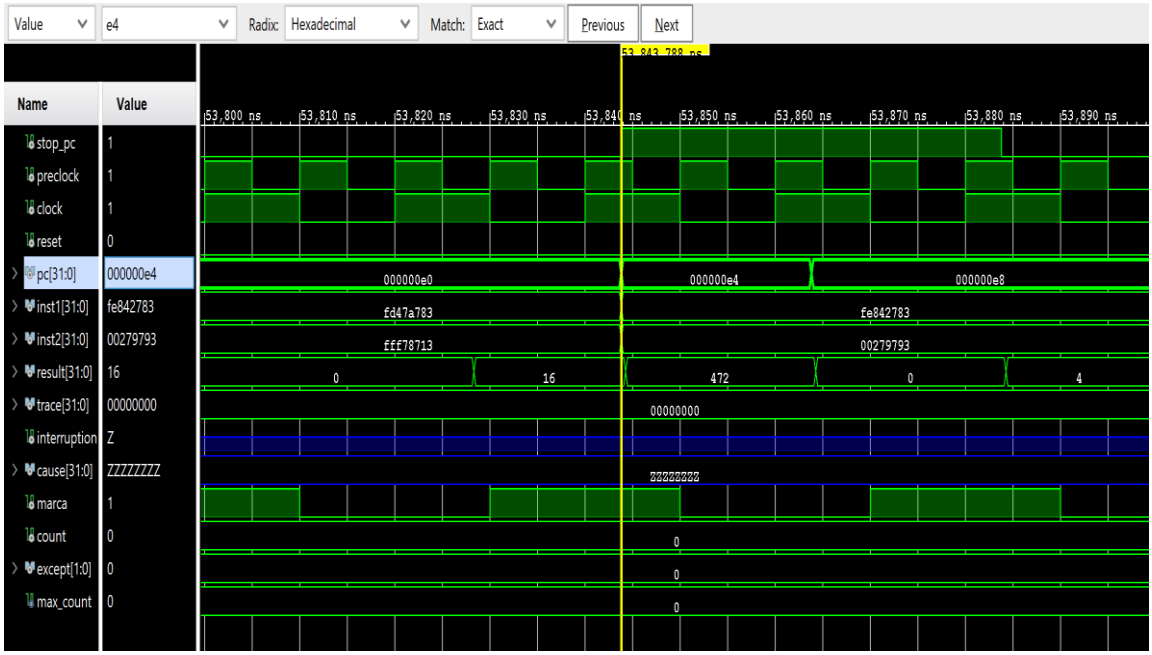


Figura 38: Simulación - PC = e4 - Result = 16

- **RESULT = 8**

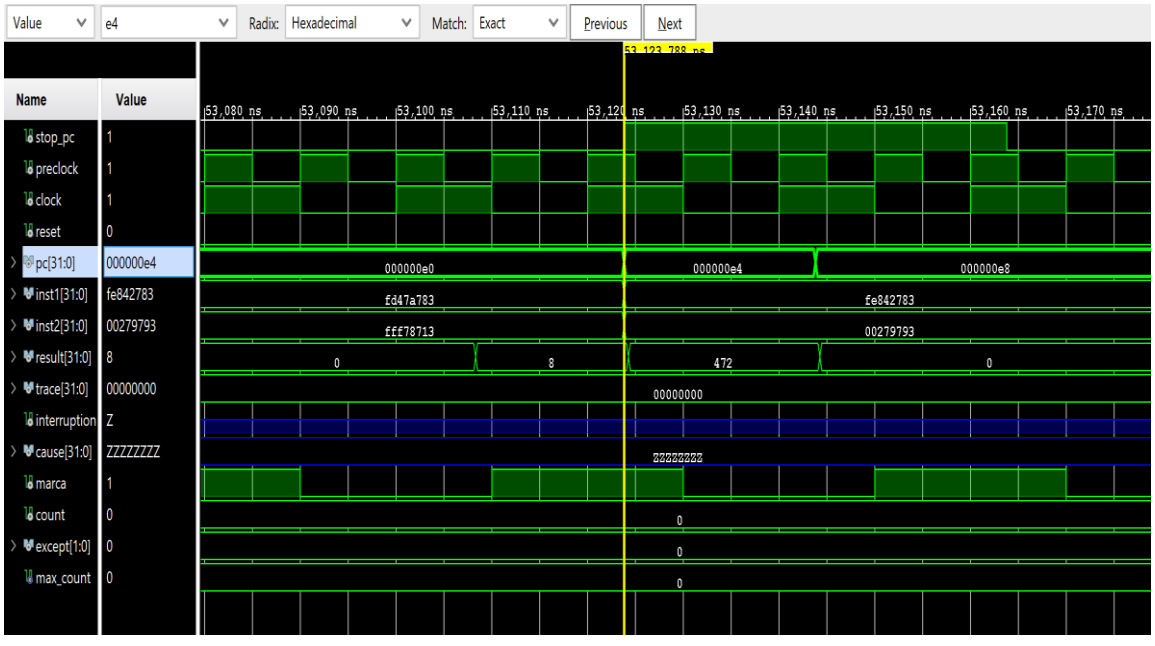


Figura 39: Simulación - PC = e4 - Result = 8

Como se ha podido observar, los resultados se han ordenado correctamente y, por lo tanto, el procesador funciona correctamente, con la mejora instalada.

3.6. IMPLEMENTACIÓN DEL PROCESADOR

Ahora que ya se ha comprobado las modificaciones realizadas en el *pipeline*, es hora de ejecutar la implementación del circuito para la posterior prueba en placa. Es necesario establecer como *Top* el *Top level* para realizar la implementación.

Hay que tener en cuenta que como se implementa el sistema completo, con el *Tilelink* incluido, los resultados del reporte son completamente diferentes a los que se obtuvieron en la síntesis. Para realizar las comprobaciones de la parte que concierne a este trabajo, es decir, el procesador RV32Xtrace, es necesario habilitar las restricciones del reloj en la entrada del procesador.

En la siguiente Figura 40, se explica cómo conseguir adaptar la señal de reloj a la entrada del procesador para poder leer los resultados que interesan y así compararlos con fidelidad. Para ello, se ha hecho clic en la barra de herramientas de Vivado en el apartado: *Tools → Timing → Edit Timing Constraints*. Una vez que aparece la ventana de *Timing Constraints*, se hace doble clic sobre el reloj que había creado con anterioridad, para ajustarlo al nuevo sistema completo.

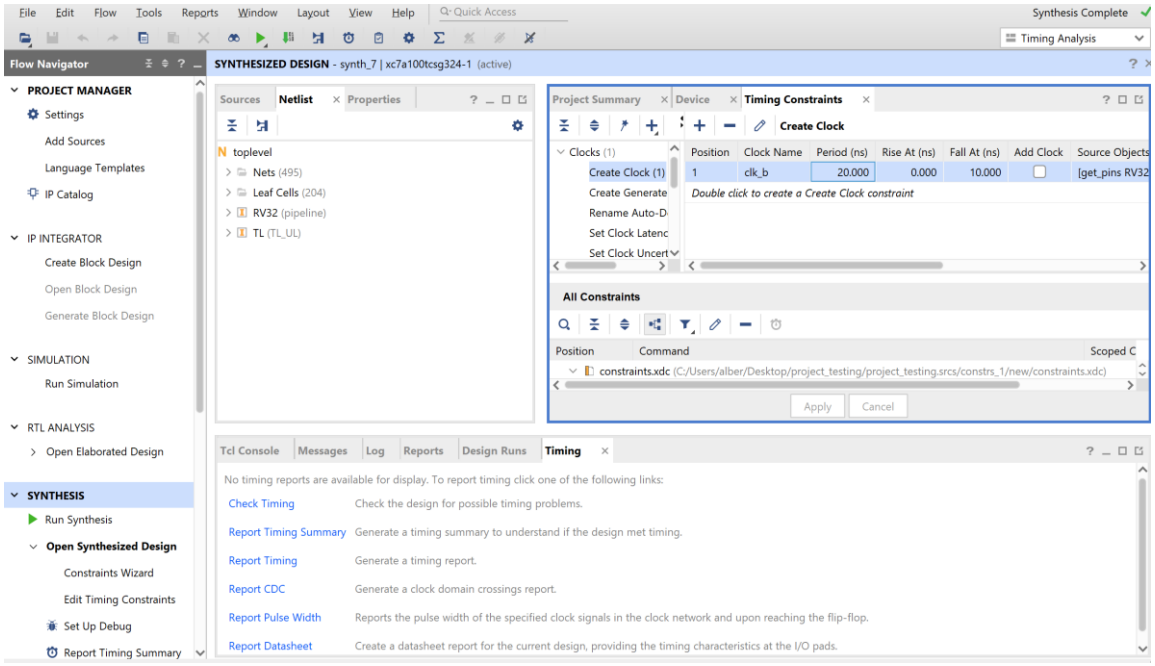


Figura 40: configuración de la señal de reloj para la implementación del sistema – Parte 1

Aparecerá una ventana como la siguiente, donde se tendrá que establecer el nombre del reloj y el origen de éste. Para establecer el origen, se hace clic en los tres puntos, tal y como indica la Figura 41.

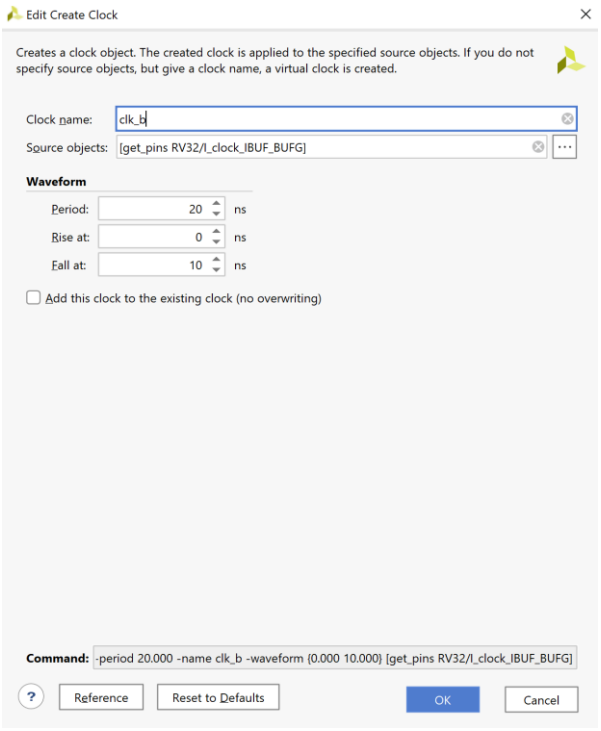


Figura 41: configuración de la señal de reloj para la implementación del sistema - Parte 2

A continuación, se busca el origen de este reloj. En el apartado *Find names of type*, se debe escoger *Cell Pins*, ya que este reloj se conecta a la entrada de la celda de entrada del reloj del procesador. En el apartado *Options*, se introduce **clock*, para filtrar la búsqueda de esta celda y en resultados se escoge la entrada al procesador, que en este caso se corresponde con la celda "*RV32/I_clock_IBUF_BUFG*". Por último, se hace clic en *Set* para guardar los cambios (ver Figura 42)

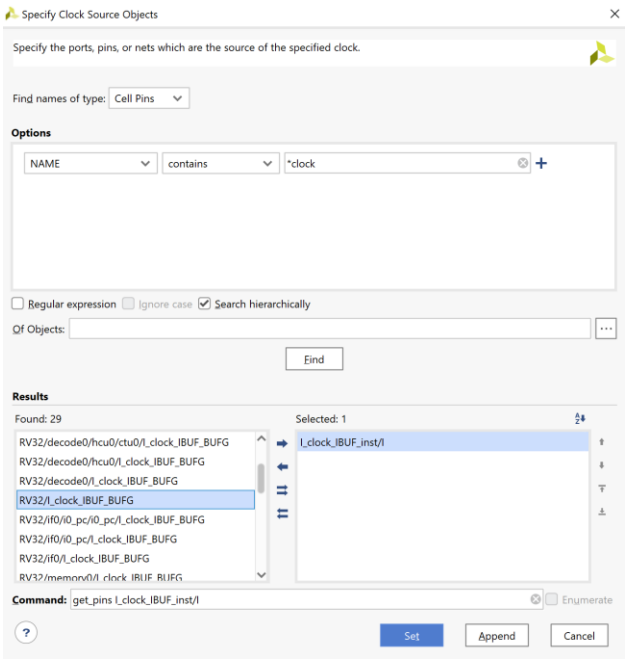


Figura 42: configuración de la señal de reloj para la implementación del sistema - Parte 3

Una vez que ya se tiene la herramienta configurada para obtener los resultados que se desean, se procede a la ejecución de la implementación. Para ello hay que dirigirse al apartado *Flow Navigator* y hacer clic en *Run Implementation* en la sección *Implementation*, tal y como indica la siguiente Figura 43.

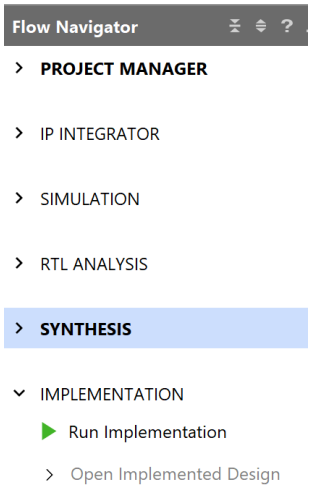


Figura 43: Ejecución de la implementación

Tal y como se ha hecho en la síntesis del circuito para sacar el reporte de tiempos, se procede de igual manera para obtener este reporte, pero esta vez en el apartado de la implementación. A continuación, se muestra el resultado que se ha obtenido mediante este reporte antes y después de la mejora (Figura 44 y Figura 45).

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Cl...
Path 1	0.409	14	107	RV32/r...[1]/C	RV32/r..._reg/D	9.610	2.956	6.654	10.0	clk_b	clk_b
Path 2	2.654	2	106	RV32/r...[0]/C	RV32/re...14]/CE	7.040	1.002	6.038	10.0	clk_b	clk_b
Path 3	2.678	2	106	RV32/r...[0]/C	RV32/re...][0]/CE	7.051	1.002	6.049	10.0	clk_b	clk_b
Path 4	2.678	2	106	RV32/r...[0]/C	RV32/re...31]/CE	7.051	1.002	6.049	10.0	clk_b	clk_b
Path 5	2.796	2	106	RV32/r...[0]/C	RV32/re...13]/CE	6.899	1.002	5.897	10.0	clk_b	clk_b
Path 6	2.796	2	106	RV32/r...[0]/C	RV32/re...27]/CE	6.899	1.002	5.897	10.0	clk_b	clk_b
Path 7	2.840	1	106	RV32/r...[0]/C	RV32/re...[26]/D	6.985	0.642	6.343	10.0	clk_b	clk_b
Path 8	2.966	2	106	RV32/r...[0]/C	RV32/re...13]/CE	6.732	0.962	5.770	10.0	clk_b	clk_b
Path 9	3.017	2	106	RV32/r...[0]/C	RV32/re...11]/CE	6.482	1.025	5.457	10.0	clk_b	clk_b
Path 10	3.017	2	106	RV32/r...[0]/C	RV32/re...15]/CE	6.482	1.025	5.457	10.0	clk_b	clk_b

Figura 44: Resultados del WNS (setup) en la implementación del sistema antes de la mejora

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Cl...	Clock Uncertainty
Path 1	1.331	11	11	64	RV32/r...g[2]/C	RV32/r..._reg/D	8.648	2.353	6.295	10.0	clk_b	clk_b	0.035
Path 2	3.313	2	3	106	RV32/r...[0]/C	RV32/re...20]/CE	6.207	0.927	5.280	10.0	clk_b	clk_b	0.035
Path 3	3.313	2	3	106	RV32/r...[0]/C	RV32/re...21]/CE	6.207	0.927	5.280	10.0	clk_b	clk_b	0.035
Path 4	3.313	2	3	106	RV32/r...[0]/C	RV32/re...24]/CE	6.207	0.927	5.280	10.0	clk_b	clk_b	0.035
Path 5	3.313	2	3	106	RV32/r...[0]/C	RV32/re...28]/CE	6.207	0.927	5.280	10.0	clk_b	clk_b	0.035
Path 6	3.314	2	3	106	RV32/r...[0]/C	RV32/r...21]/CE	6.233	0.926	5.307	10.0	clk_b	clk_b	0.035
Path 7	3.314	2	3	106	RV32/r...[0]/C	RV32/r...24]/CE	6.233	0.926	5.307	10.0	clk_b	clk_b	0.035

Figura 45: Resultados del WNS (setup) en la implementación del sistema después de la mejora

El peor caso de ejecución tanto antes de la mejora como después de la mejora pertenece a la ruta que hacen las instrucciones de salto condicional. Antes de la mejora, al ejecutar la implementación del sistema, se ha obtenido un resultado **WNS = 0.409 ns**, mientras que después de la mejora, este resultado ha sido **WNS = 1.331 ns**, comprobándose una clara mejora en la velocidad del procesador.

Después de comprobar que los resultados de la implementación han mejorado con respecto a los obtenidos antes de la mejora, se han realizado de nuevo las simulaciones, aunque esta vez en post-implementación. Los resultados obtenidos han sido satisfactorios, comprobándose de nuevo que los números se han ordenado con éxito. Como este resultado es el mismo que se obtiene en simulación post-síntesis, no se han adjuntado capturas, ya que son las mismas que el anterior apartado.

Para finalizar, en el siguiente apartado, se muestra la realización de las pruebas en placa para hacer una comprobación práctica de la funcionalidad del procesador y de demostrar que la mejora realizada funciona correctamente.

3.7. PRUEBA EN PLACA

Para concluir la parte experimental de la mejora realizada en el procesador, se han realizado pruebas en la placa Nexys 4 DDR, obteniendo con ello las comprobaciones pertinentes para demostrar que las modificaciones realizadas funcionan de manera correcta.

Lo primero que se necesita hacer es conectar la placa al ordenador para que sea leído por la herramienta. Se conectará el cable al puerto USB del ordenador y al puerto UART de la placa. Con la placa conectada, ya podemos generar el *Bitstream* desde Vivado. Para ello hay que dirigirse a *Flow Navigator* y hacer clic en *Generate Birstream* en la sección *Program and Debug* como indica la siguiente Figura 46.

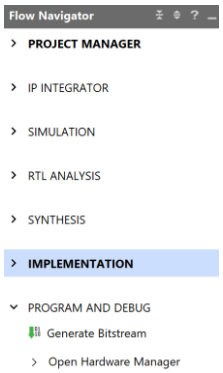


Figura 46: Generate Bitstream

Una vez generado el *bitstream* y conectado la placa, aparecerá la siguiente vista de Vivado (Figura 47). A continuación, hay que programar la placa introduciendo el *bitstream* y el archivo de depuración. En la Figura 48 aparece automáticamente estos dos ficheros por defecto. En este punto, se hace clic en *Program*, e inmediatamente la placa cargará el programa para poder ejecutarlo.

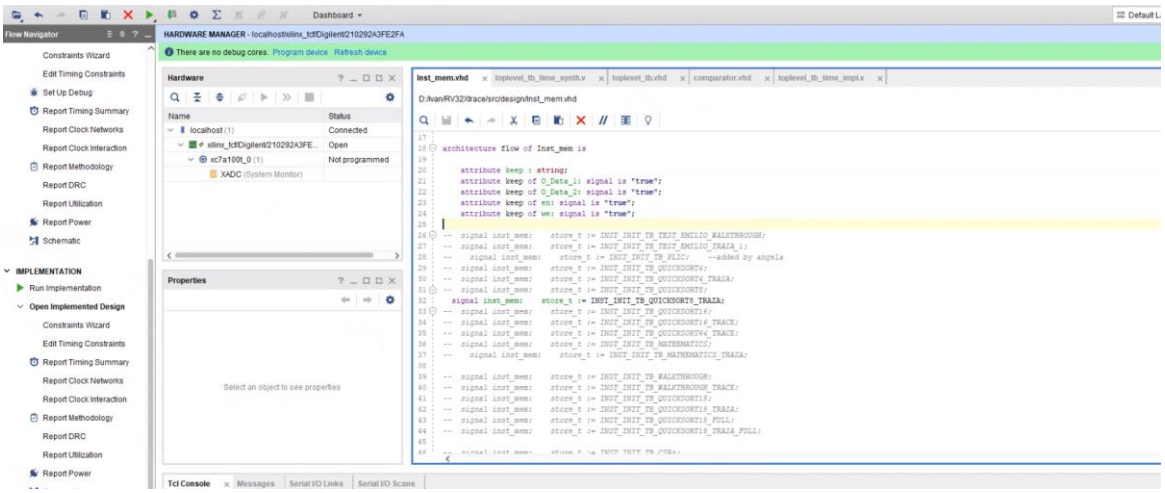


Figura 47: Depuración del programa en la placa

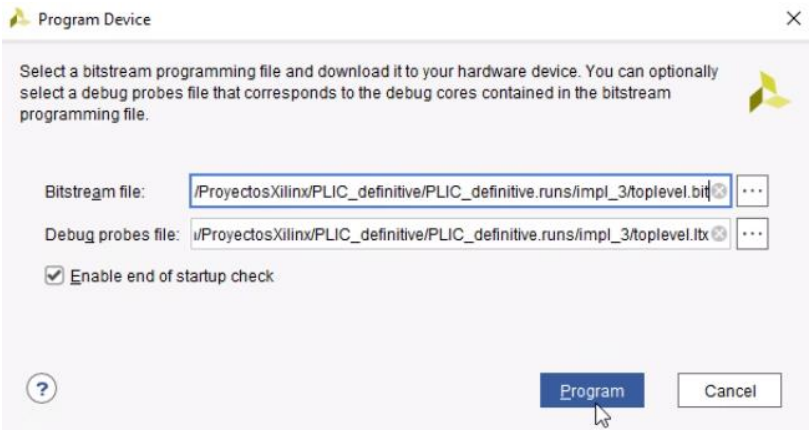


Figura 48: Elección de archivos para la depuración

En la siguiente Figura 49 se puede observar la placa utilizada en este trabajo, mostrándose con el programa cargado. A simple vista en la placa, no se puede apreciar si el programa se está ejecutando bien o no, ya que al trabajar a una frecuencia alta (50 MHz) y tener los leds habilitados como señales de salida, no dan un claro resultado. Por lo tanto, se ha configurado un analizador lógico para comprobar los resultados obtenidos de manera visual.

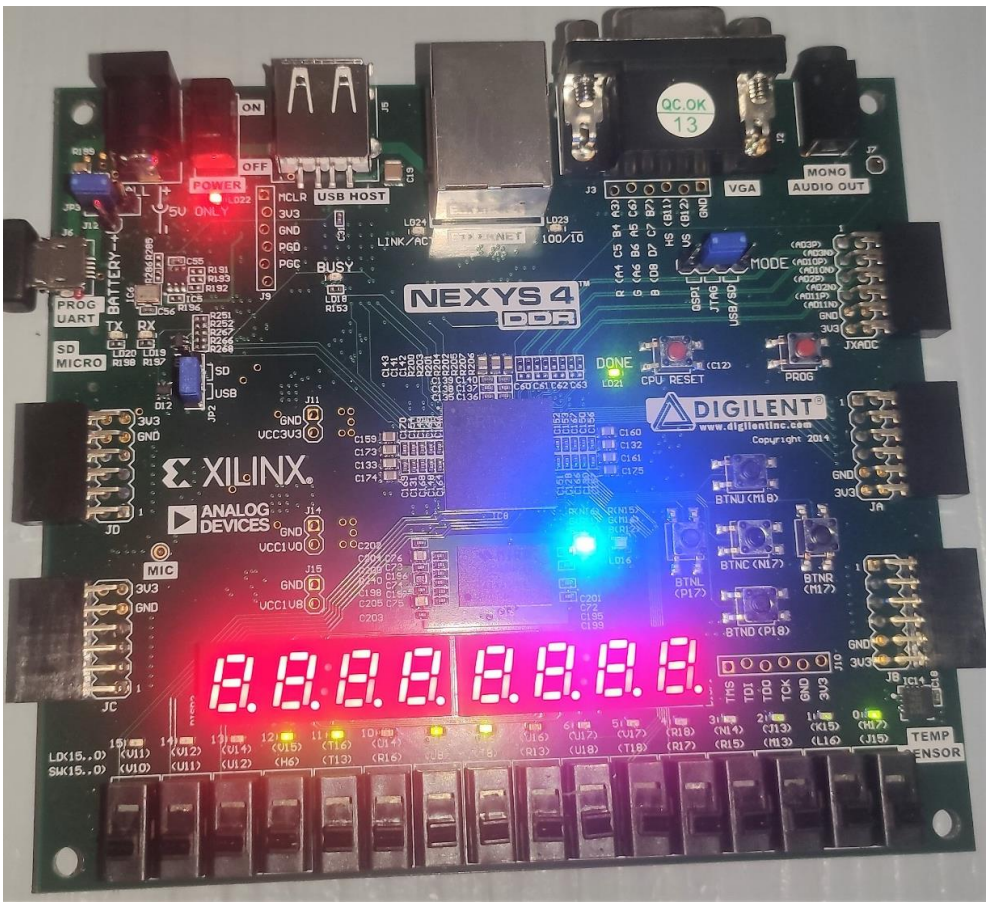


Figura 49: Placa Nexys 4 DDR

Una vez se ha hecho clic en *Program* para cargar el programa en la placa, en Vivado aparece la siguiente Figura 50, donde se va a poder comprobar si el procesador se está ejecutando correctamente. Se ha utilizado por parte del diseñador del procesador para ello, un analizador lógico integrado (*ILA*), de Xilinx, para así analizar los resultados que se obtienen al ejecutar el procesador en la placa.

En la configuración de la *ILA* se pueden configurar la cantidad de muestras que se quieren obtener, para comprobar el funcionamiento del procesador. En este caso se ha establecido 8,192 muestras y se ha elegido la señal de *reset*, es decir, el botón *reset* de la placa, para activar el procesador.

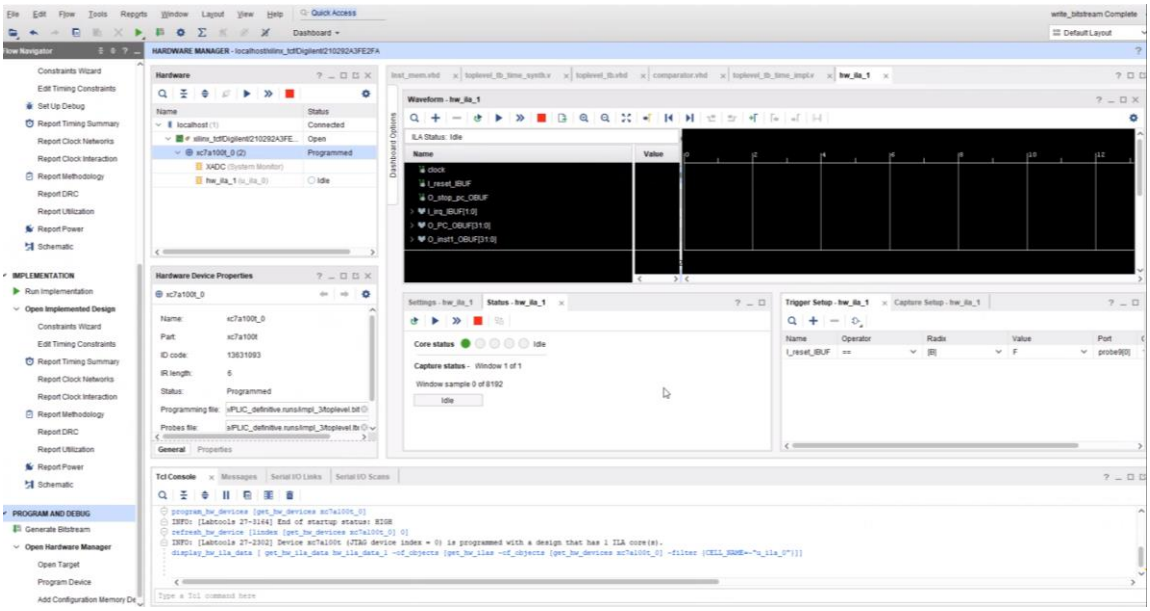


Figura 50: Analizador lógico - Configuración inicial

Una vez se pulsa el botón *reset*, y justo antes de soltarlo empieza a coger ciertas muestras, que son las correspondientes con la configuración del *trigger*, en este caso 30 muestras, es decir, el programa se mantiene con su estado inicial durante 30 muestras para después poder ejecutarse, es decir, como un disparador. La marca roja de la Figura 51 muestra las 30 muestras cogidas antes del *trigger* y la marca amarilla indicaría el inicio del programa con “PC = 0”.

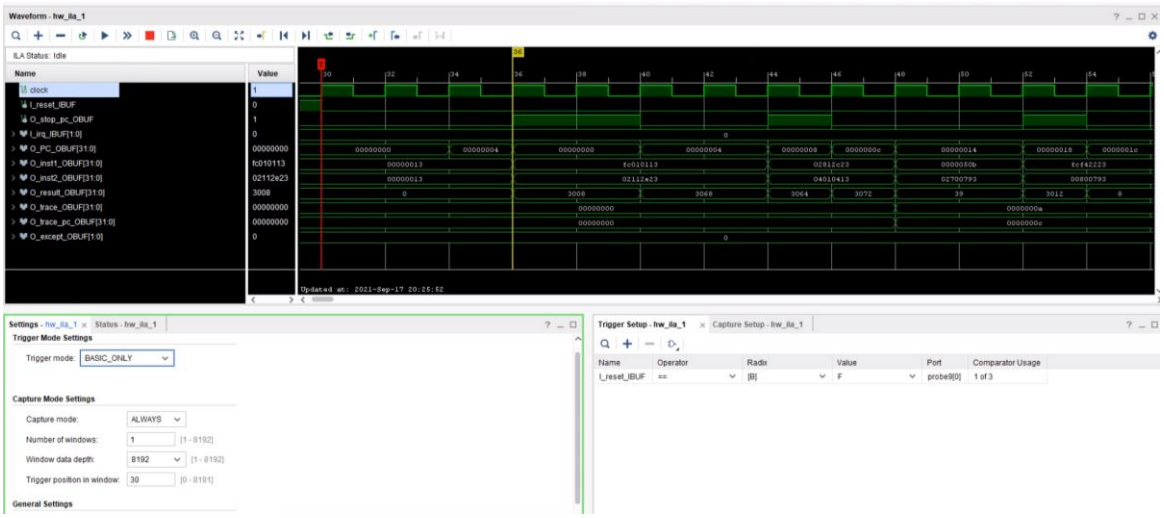


Figura 51: Analizador lógico - Trigger y PC = 0

Por último, se ha comprobado que el procesador funciona de manera correcta, comprobando que el programa que se ha ejecutado para ordenar una serie de números se ha ejecutado correctamente. En la siguiente Figura 52, como se realizó en las simulaciones, se ha buscado el "PC = e0", que es donde se obtiene la señal de salida *result*, y se comprueba que el resultado en esta captura es "*result* = 8".

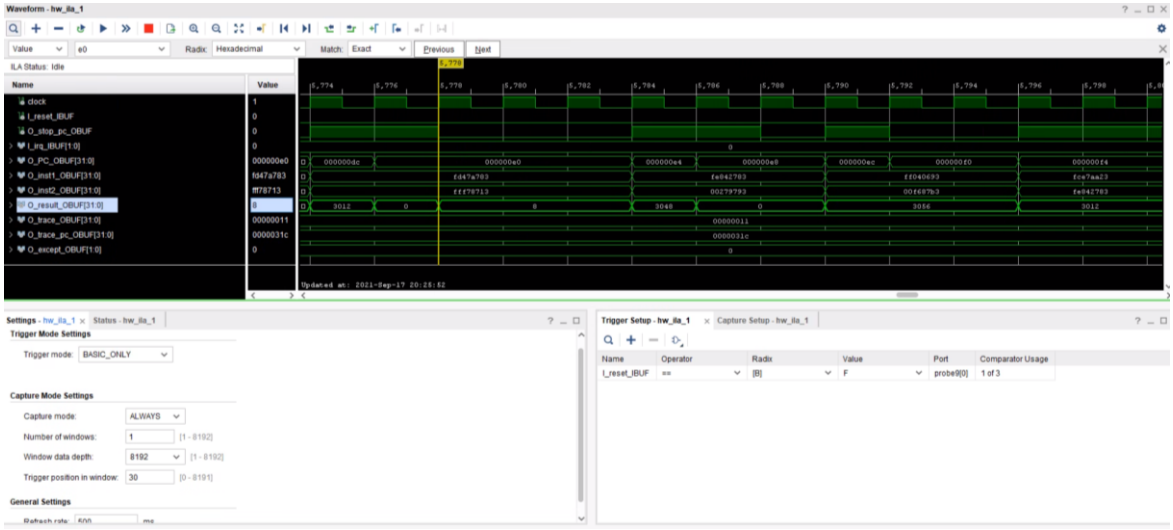


Figura 52: Analizador lógico - Result = 8

4. CONCLUSIÓN Y FUTURAS PROPUESTAS

En este trabajo se ha realizado el estudio teórico y el análisis para caracterizar un camino crítico que condicionaba la frecuencia del procesador. A continuación, se ha realizado una modificación sobre el mismo, para conseguir una mayor holgura de tiempo, y así poder aumentar la frecuencia de trabajo del procesador.

Como se ha comprobado al realizar las pruebas con todo el sistema completo, no se ha podido aumentar la frecuencia del sistema, ya que la parte que lo ralentiza no está dentro del procesador en sí, sino que se encuentra en otros componentes externos al procesador, los cuales también están en desarrollo por parte de otros proyectos paralelos a éste.

La mejora que se ha realizado es una de las muchas que podría haber, ya que siempre habrá un camino crítico sobre el que trabajar. Como posibles mejoras, se podría optimizar el hardware, o conseguir que la separación entre componentes sea menor, para así disminuir el tiempo entre los nodos, o modificar la estructura de las etapas, para conseguir optimizar la lógica de la ruta, etc.

ANEXO 1: INSTALACIÓN DE VIVADO2018.3

En este anexo se explica paso a paso la instalación del programa utilizado para el diseño, simulación e implementación del procesador en RISC-V.

Para la instalación de Vivado, lo primero que se necesita es un equipo con conexión a Internet y con un sistema operativo compatible con la herramienta, como puede ser Windows o Linux. En este trabajo se ha desarrollado la instalación sobre un equipo con Windows como sistema operativo.

Para la instalación del programa hay que dirigirse a la página web del proveedor del programa, en este caso, Xilinx, en el enlace https://www.xilinx.com/support/download.html. En el apartado de descargar [7] se debe seleccionar la versión de Vivado 2018.3 (Figura 53), ya que es sobre la que se ha desarrollado el procesador y así se evitarán errores de actualización.

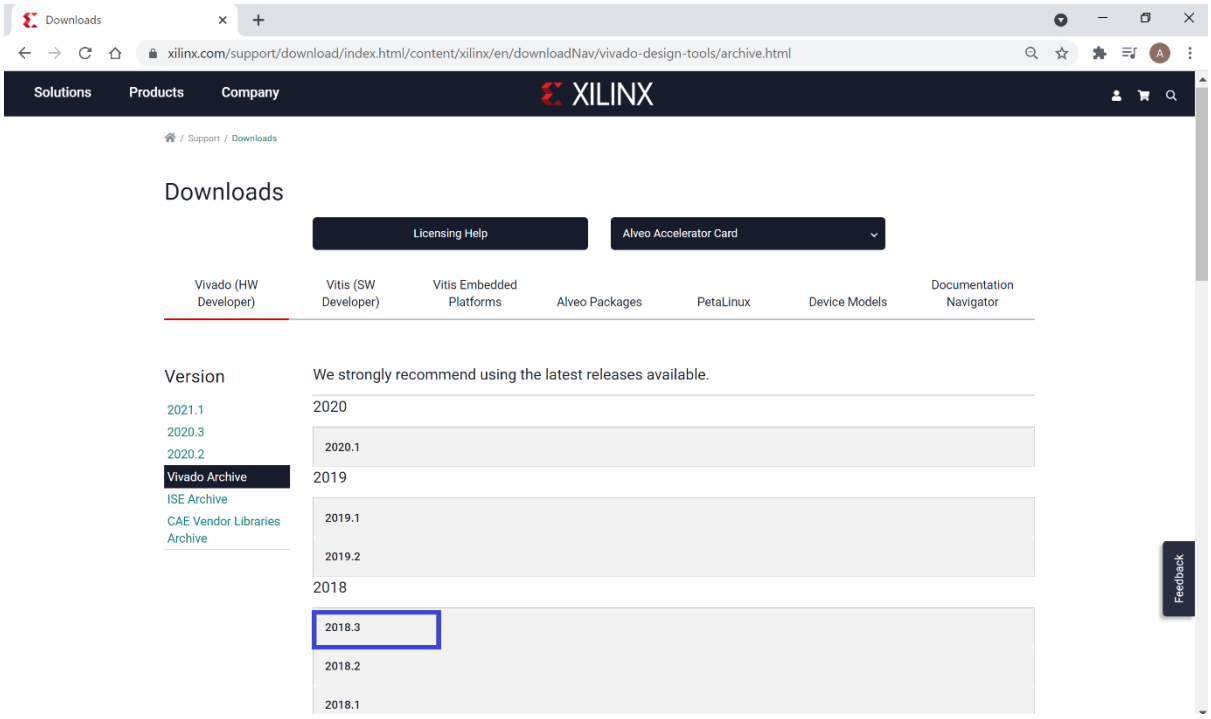


Figura 53: Web Xilinx - Descarga Vivado – Versión 2018.3

Una vez que se ha encontrado la versión de Vivado a instalar, se hará clic sobre ella y se mostraran las distintas opciones de descarga. Buscar la mejor opción que se adapte al equipo/sistema operativo que se utilizará y hacer clic en el enlace de descarga, tal y como se muestra en la siguiente Figura 54. En este caso se ha escogido la opción "*VivadoHLx 2018.3: WebPACK and Editions – Windows Self Extracting Web Installer*".

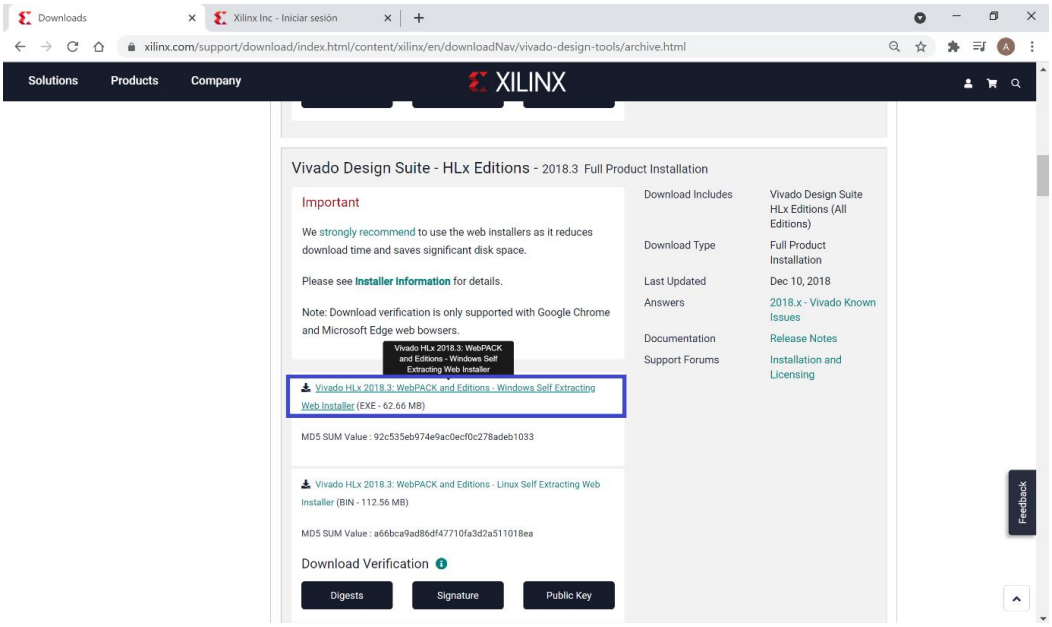


Figura 54: Web Xilinx - Descarga Vivado – Versión – Instalador

Cuando se accede al enlace de descarga, se abre automáticamente una pestaña para iniciar sesión con una cuenta de Xilinx. En caso de no tener una cuenta, se podrá crear una nueva rellenando un formulario con los datos personales.

Al iniciar sesión, la web redirige al apartado de la cuenta personal donde se pueden comprobar los datos personales y, después de comprobarlo se procederá a la descarga haciendo clic en *Download* (Figura 55).

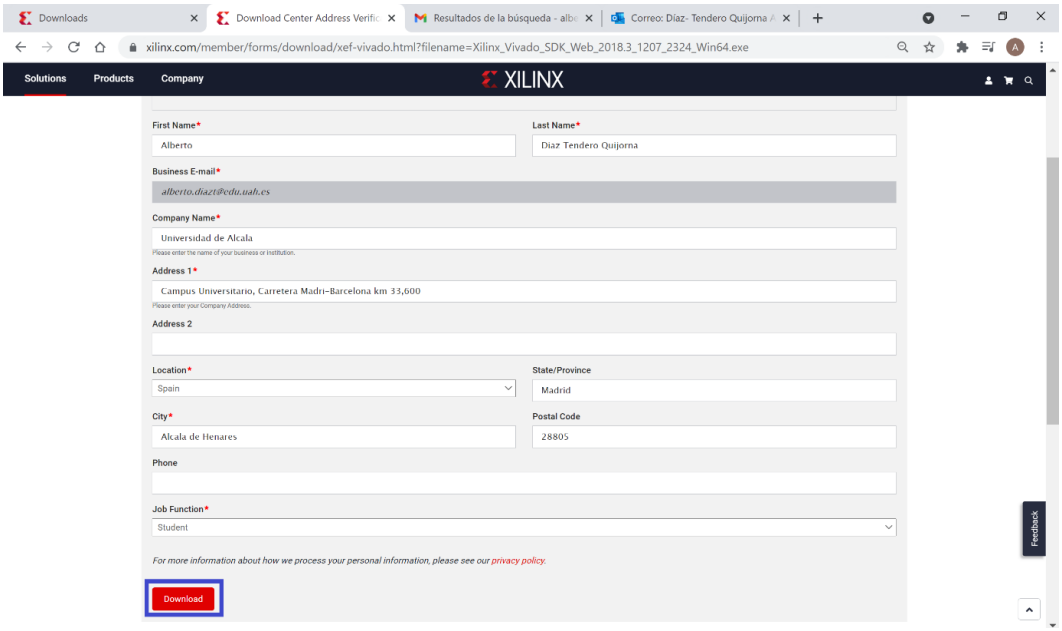


Figura 55: Web Xilinx - Descarga Vivado – Cuenta personal

Una vez descargado el instalador, se tiene que ejecutar para la instalación. Pasados unos segundos pregunta si se quiere instalar la última versión de Vivado, a lo que se debe omitir, haciendo clic en Continuar (Figura 56). Posteriormente hacer clic en *Next*, tal y como se muestra en la siguiente Figura 57Figura 58:

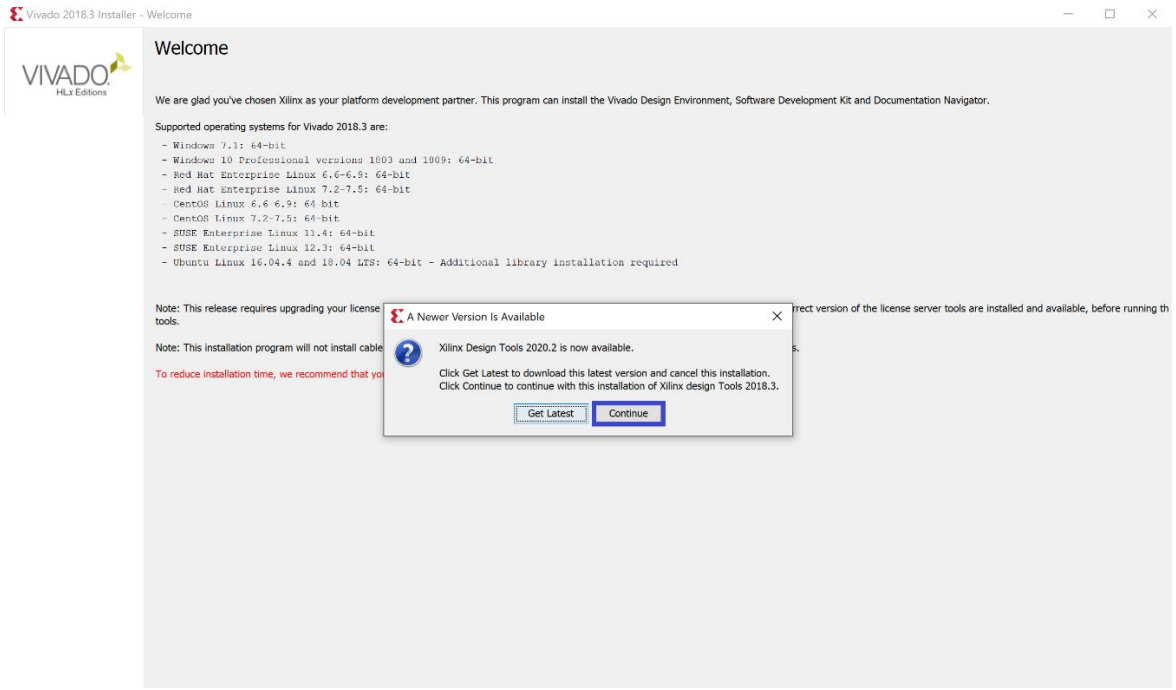


Figura 56: Descarga Vivado – Instalador - Omitir descargar nueva versión

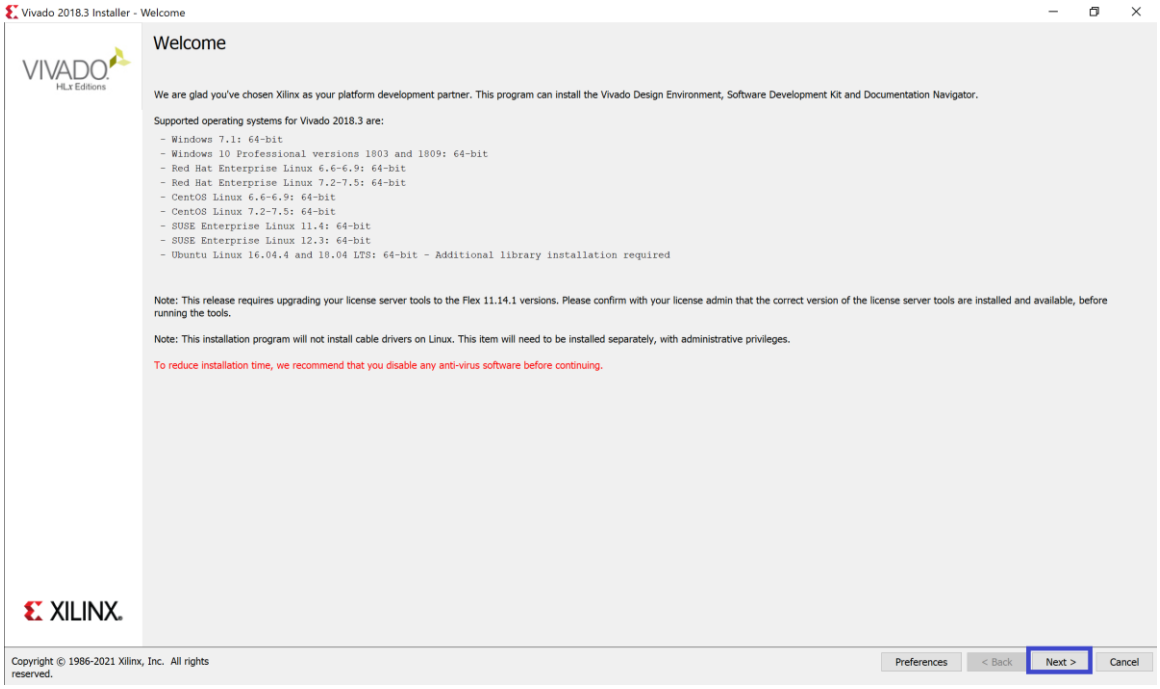


Figura 57: Descarga Vivado – Instalador - Welcome

En la siguiente sección, se preguntan las credenciales de usuario y contraseña de la cuenta personal. Además, se debe marcar la opción *Download and Install now*. Una vez hecho esto, hacer clic en *Next* (Figura 58).

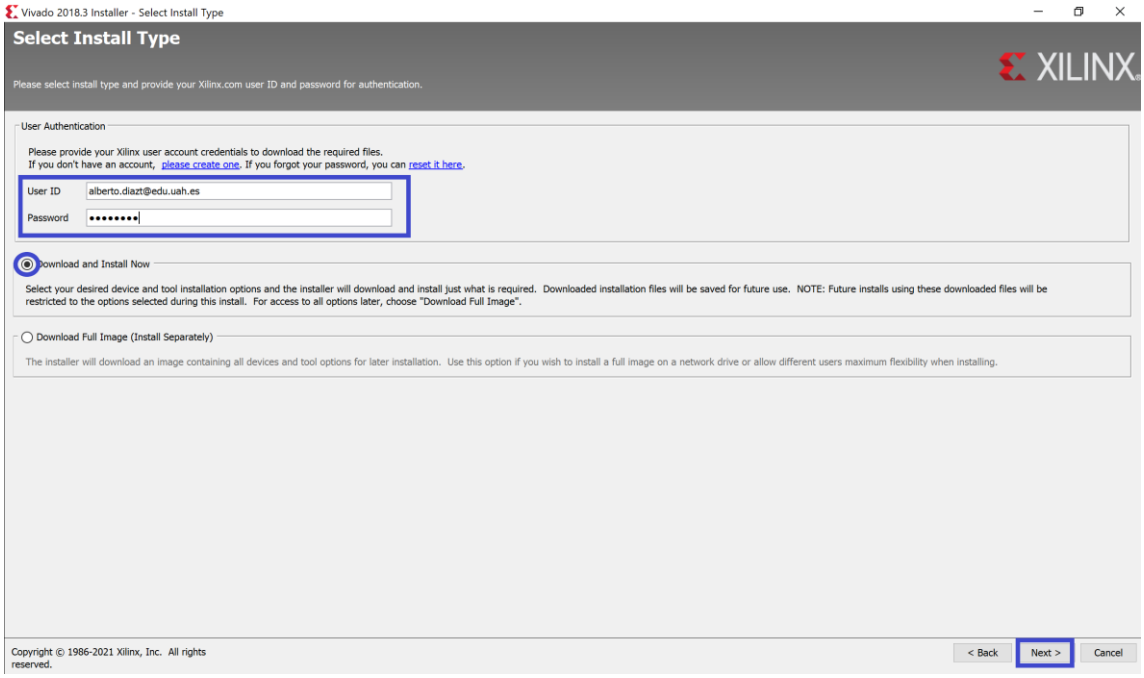


Figura 58: Descargar Vivado – Instalador – Credenciales

Confirmar los términos y condiciones haciendo clic en las tres casillas *I Agree*. Hacer clic en *Next* (ver Figura 59)

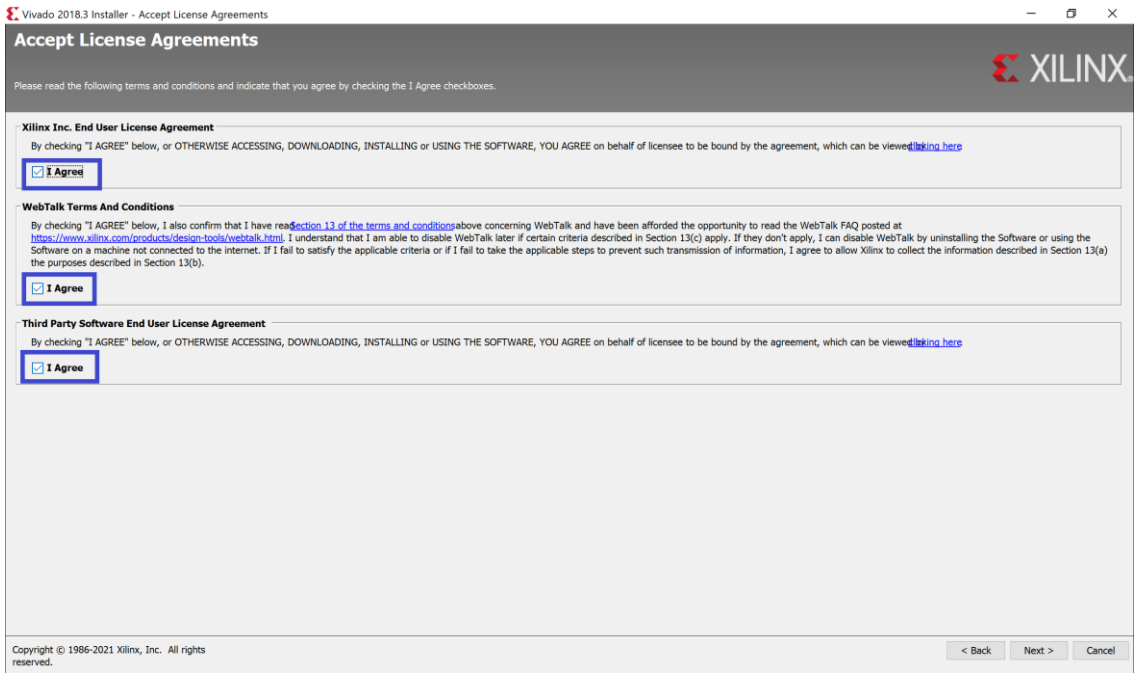


Figura 59: Descarga Vivado – Instalador - Aceptación de acuerdos de licencia

A continuación, se debe seleccionar la edición de Vivado que se desea instalar. En este caso, seleccionar la opción *Vivado HL Design Edition* y hacer clic en *Next* (Figura 60).

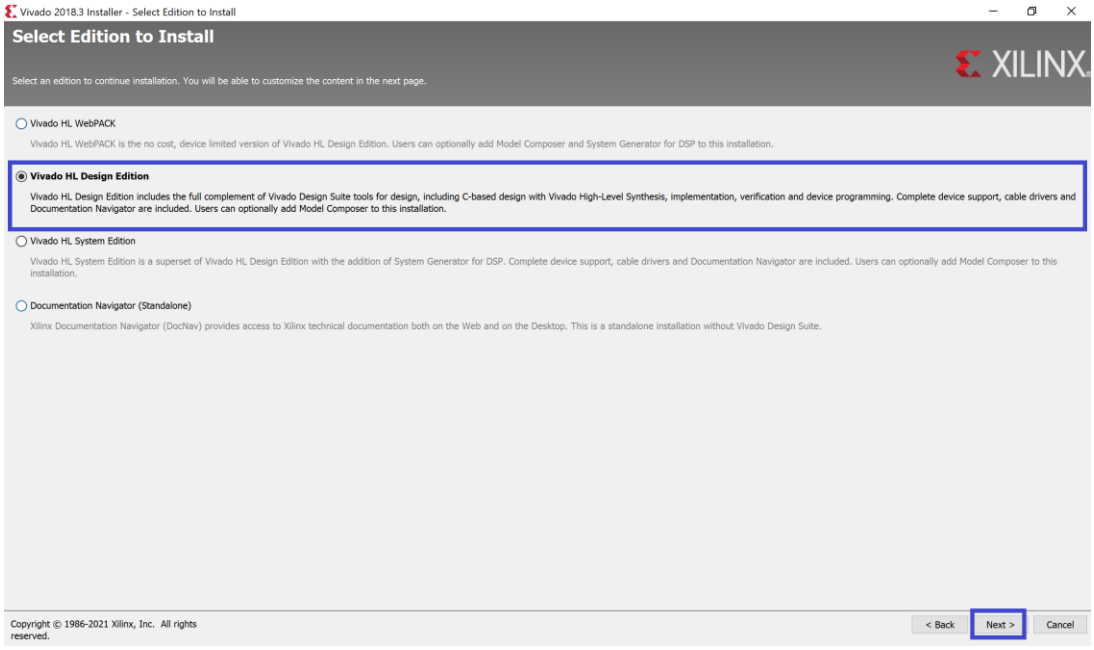


Figura 60: Descarga Vivado – Instalador - Selección de la edición de Vivado

En la siguiente pantalla se elige la configuración de la instalación del programa. En este caso, se dejará todo por defecto y se hace clic en *Next* (Figura 61).

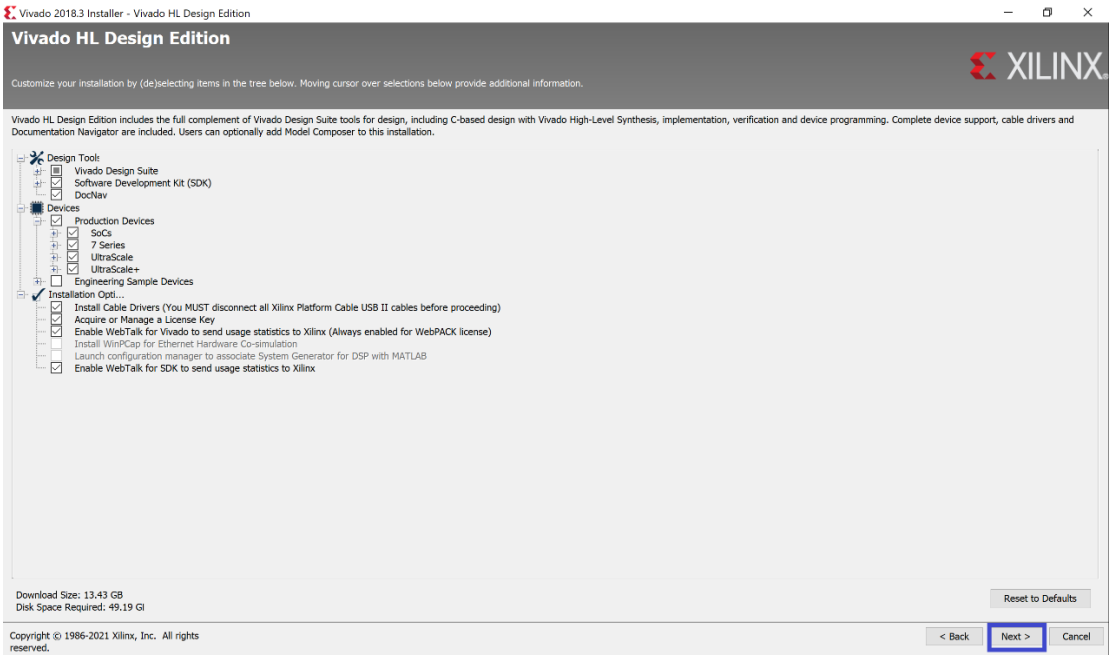


Figura 61: Descarga Vivado – Instalador - Customización de la versión de vivado

Una vez se ha elegido la configuración de Vivado, hay que elegir el directorio donde se instalará el programa. Lo dejaremos por defecto en el directorio creado en C:/Xilinx. También aparecen en esta sección otros directorios de instalación dentro del directorio de Xilinx, donde se instalará *Vivado*, *SDK* y *DocNav*. Por último, también se indica el tamaño de la instalación, además del espacio requerido y del espacio disponible. Una vez realizado este paso, hacer clic en *Next* (Figura 62).

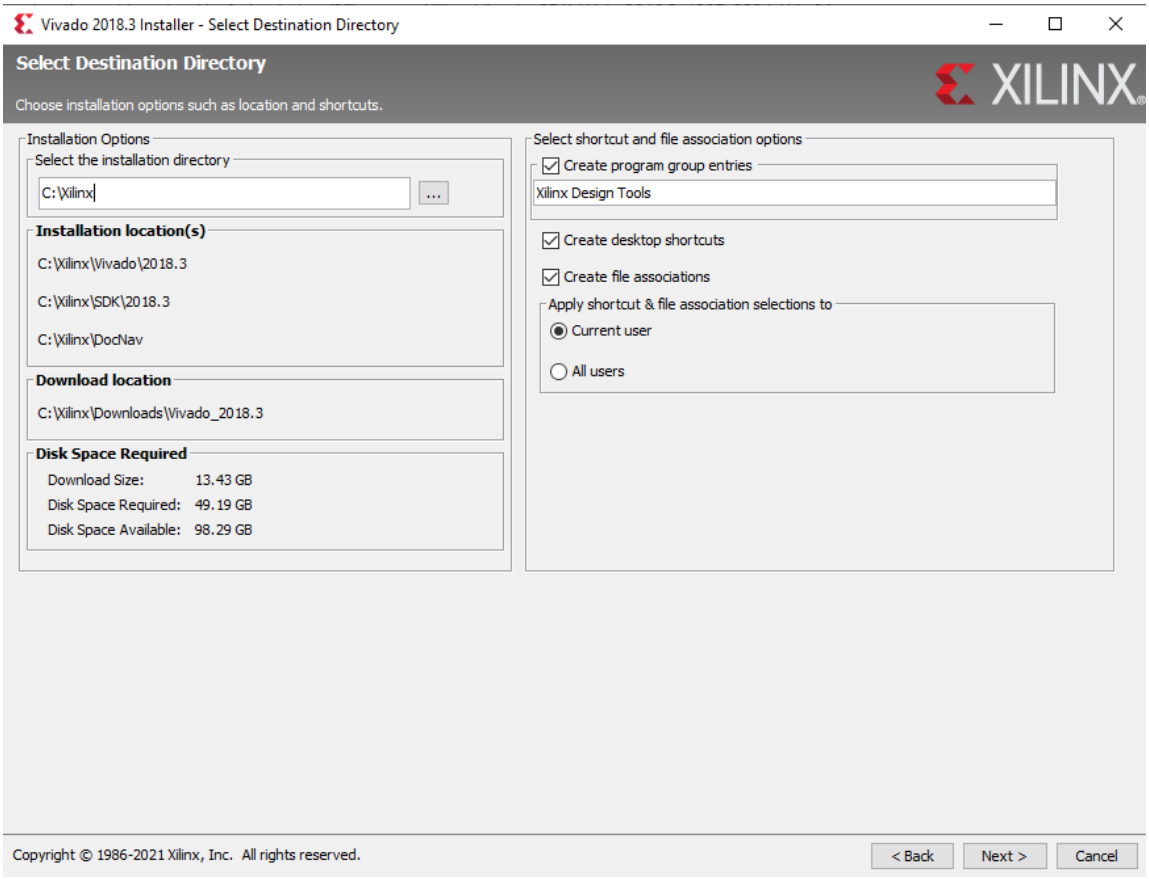


Figura 62: Descarga Vivado – Instalador - Selección de directorio

Antes de proceder a la instalación en el disco del sistema, aparece un resumen de la instalación, donde se pueden observar las opciones que se han elegido anteriormente. Se debe hacer clic en *Install* para comenzar la instalación, tal y como se muestra en la Figura 63. Después aparecerá la ventana del proceso de la descarga, que requerirá algo más de tiempo en completarse (Figura 64).

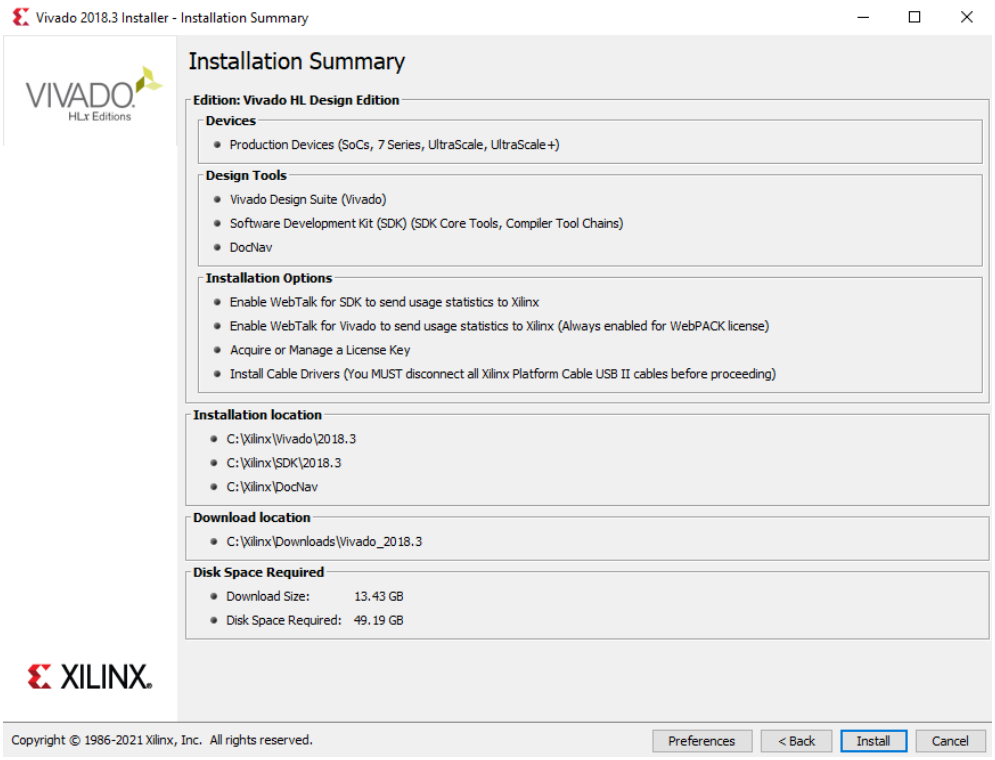


Figura 63: Descarga Vivado – Instalador - Resumen de la instalación

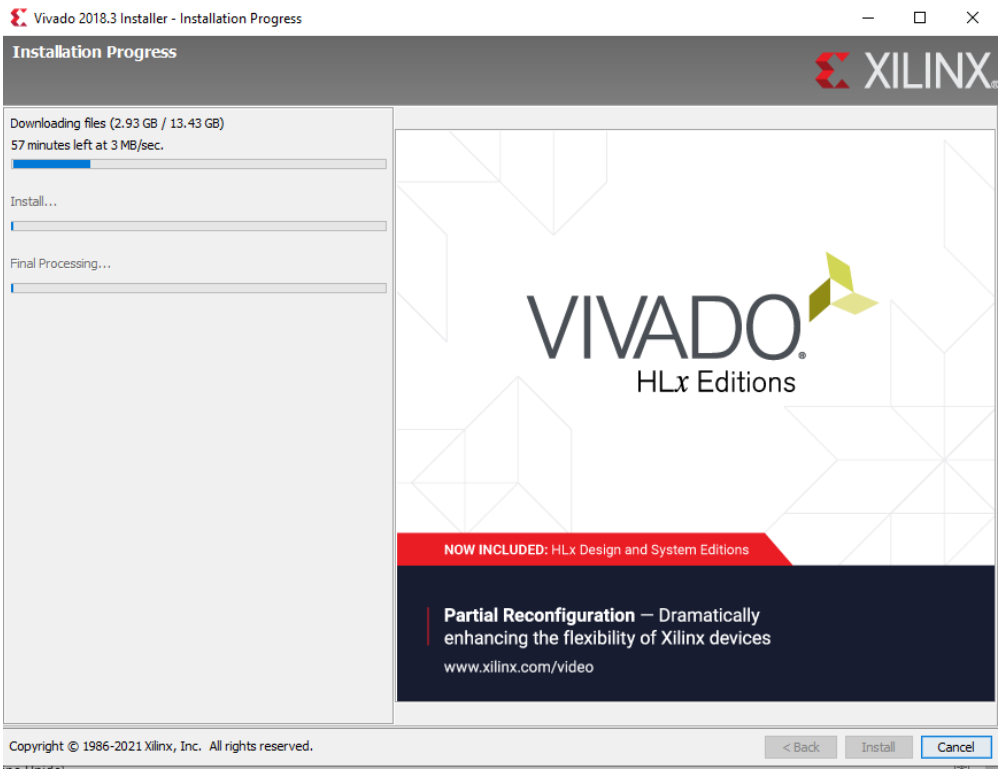


Figura 64: Descarga Vivado – Instalador – Progreso – Parte 1

Una vez completada la instalación, aparecerá un mensaje diciendo que la instalación se ha completado con éxito, como se puede ver en la siguiente Figura 65.

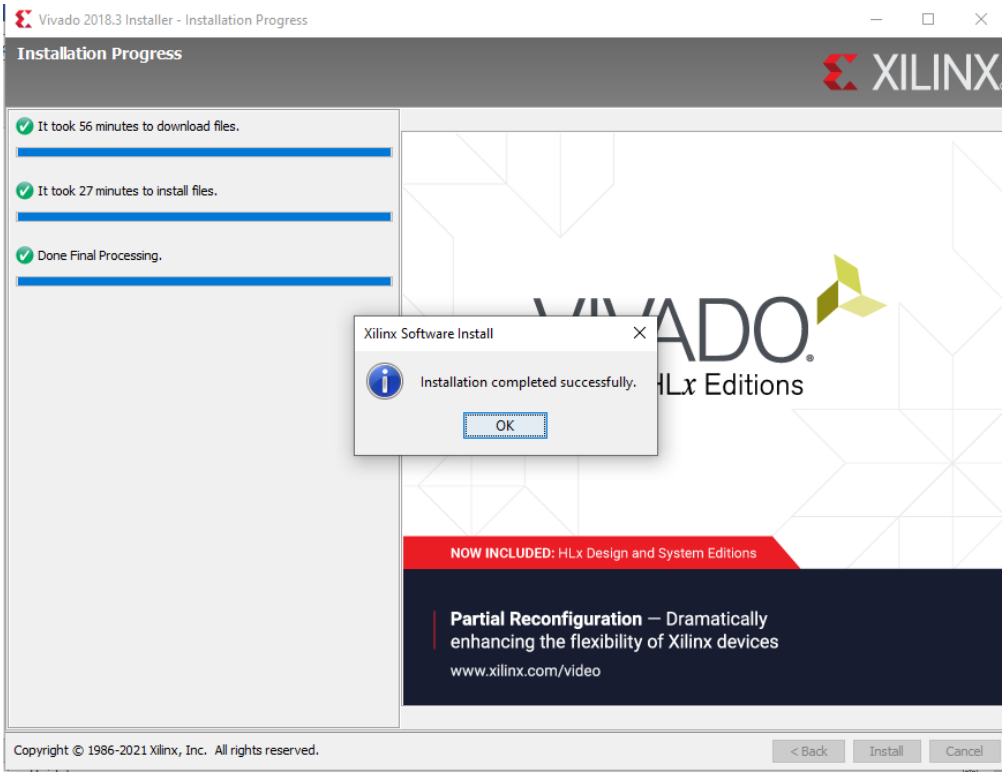


Figura 65: Descarga Vivado – Instalador - Progreso - Parte 2

A continuación, aparece el gestor de licencias de Vivado. En este momento se necesita obtener las licencias para incluirlas en nuestro sistema y así tener el acceso a la herramienta. Para ello, debemos seleccionar en la parte de la izquierda de la ventana *Obtain License* y dentro de este apartado la opción *Get My Full or Purchased Certificate-Based License*. Una vez seleccionada esta opción, hacemos clic en *Connect Now*, tal como indica la Figura 66.

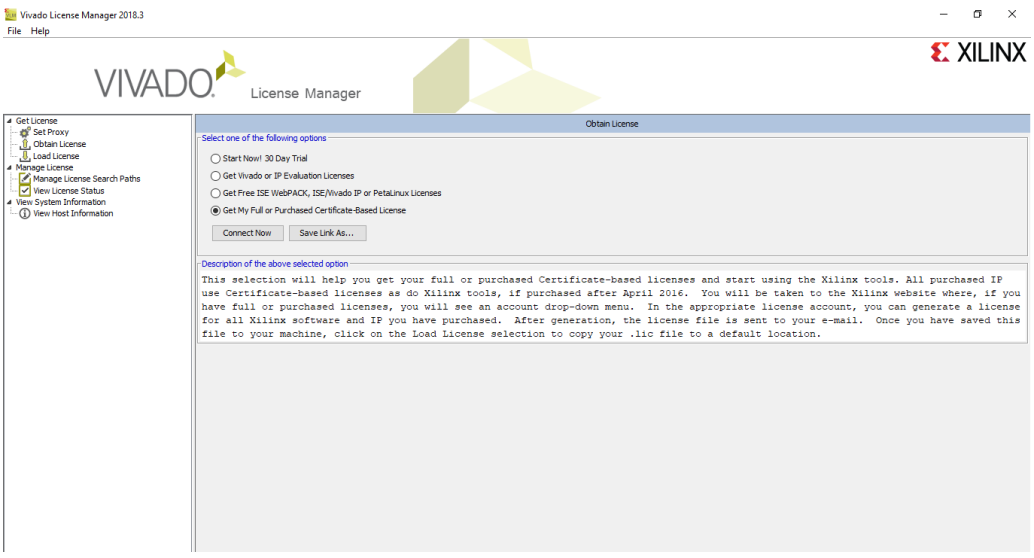


Figura 66: Descarga Vivado – Licencia - Parte 1

En este momento se abrirá automáticamente una ventana del navegador que dirige a la página de Xilinx, donde se rellenarán los datos personales de la cuenta, en caso de que no estén ya rellenados y se pulsará el botón *Next* (Figura 67).

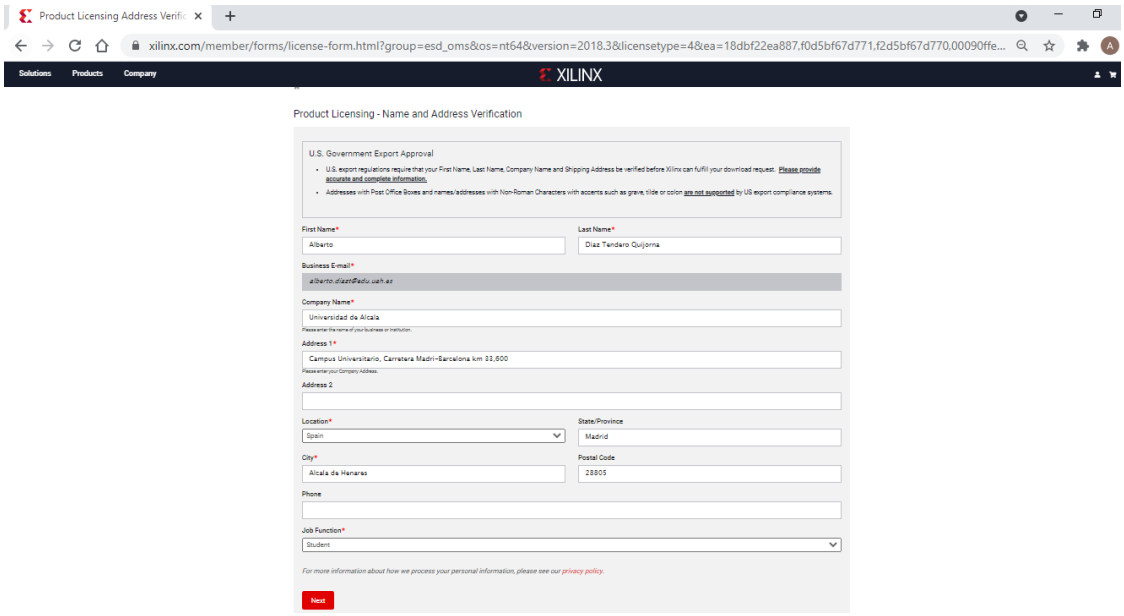


Figura 67: Descarga Vivado – Licencia - Parte 2

Si todo es correcto, la página web se actualizará dirigiéndose al apartado para crear una nueva licencia y aparecerá una lista en la que se encuentra la opción: *Vivado Design Suite: HL WebPACK 2015 and Earlier License*. Debemos seleccionarla y pulsar sobre el botón *Generate Node-Locked License*, como se muestra en la Figura 68.

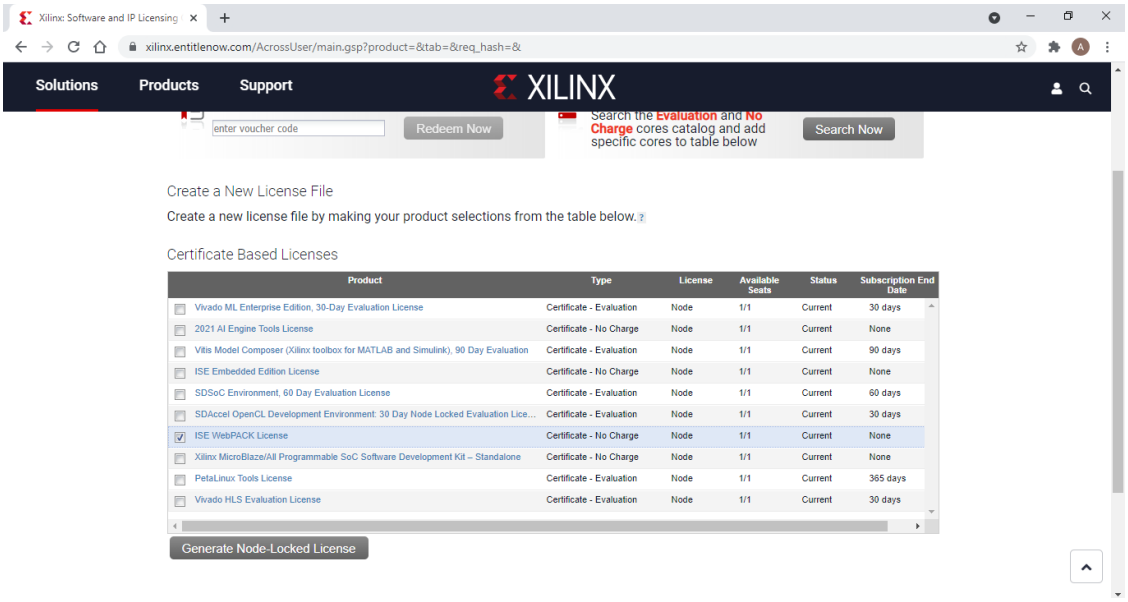


Figura 68: Descarga Vivado – Licencia - Parte 3

Una vez generada la licencia, es importante comprobar en el navegador que está generada correctamente. Para ello, hay que dirigirse al apartado *Manage Licenses*, comprobar que la licencia que se ha seleccionado es la que reside en el repositorio y descargar la licencia, tal y como se aprecia en la Figura 69.

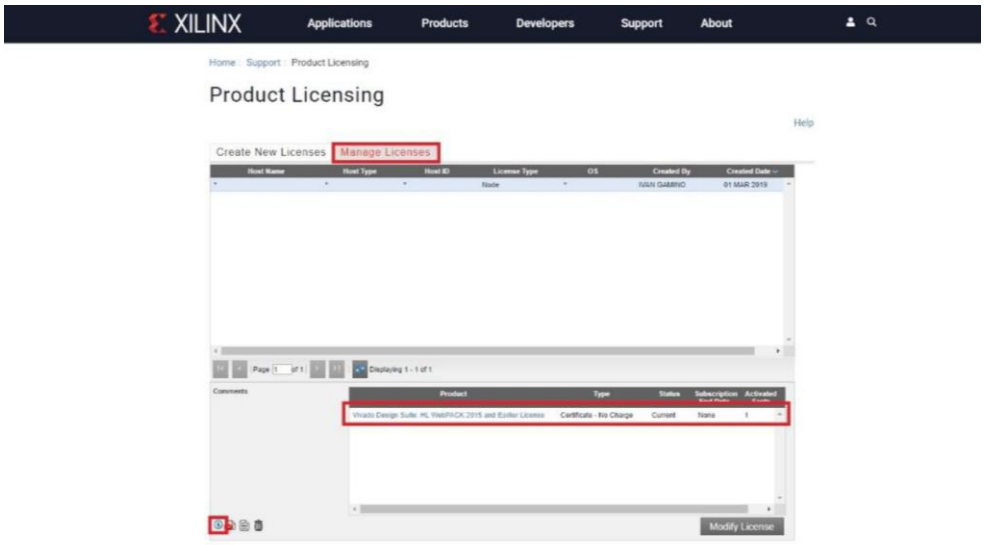


Figura 69: Descarga Vivado – Licencia - Parte 4

Una vez completada la descarga de la licencia, es necesario incluirla en el gestor de licencias de Vivado, cuya ventana estaba abierta anteriormente. En el apartado a la izquierda de la ventana, se selecciona *Load License* y a continuación, *Copy License*…. En la Figura 70 se pueden visualizar estos pasos.

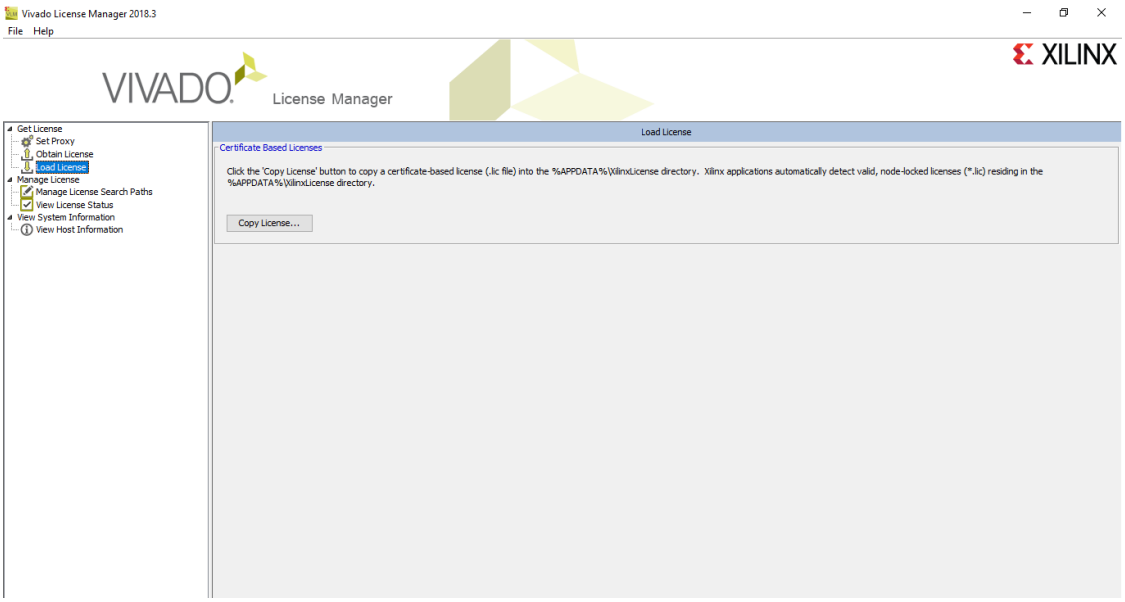


Figura 70: Descarga Vivado – Licencia - Parte 5

Se abrirá a continuación una ventana donde se debe localizar el archivo de licencia que se ha descargado para cargarlo en el gestor de licencias, así como se puede ver en la Figura 71. Una vez completado este proceso de manera correcta, aparecerá un mensaje indicando que se ha instalado la licencia con éxito, tal y como indica la Figura 72.

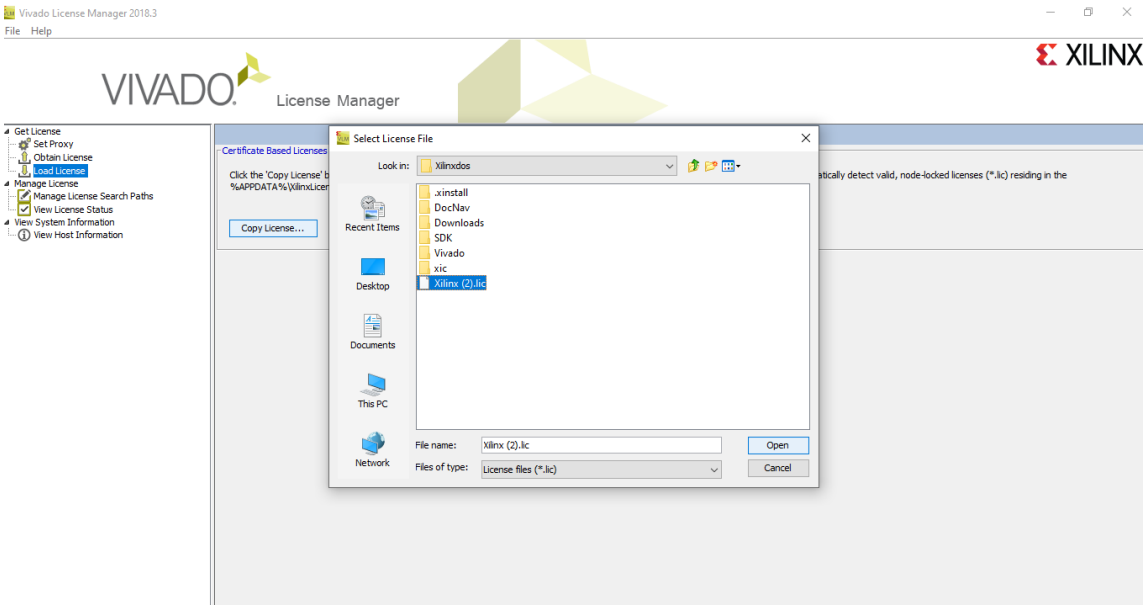


Figura 71: Descarga Vivado – Licencia - Parte 6

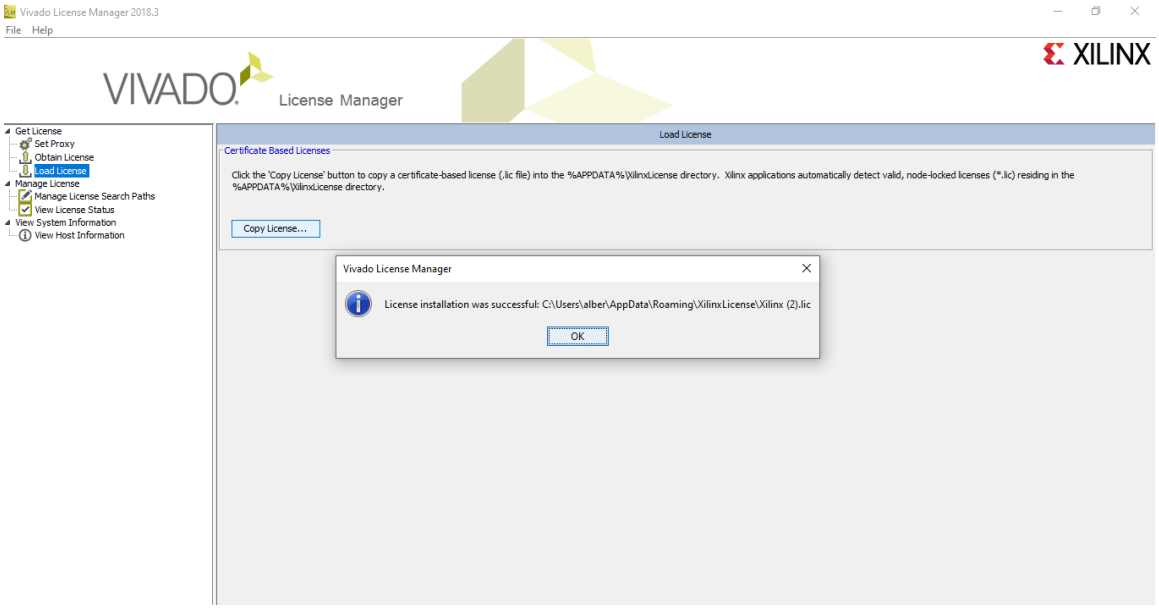


Figura 72: Descarga Vivado – Licencia - Parte 7

Para comprobar que se ha cargado la licencia correctamente, debemos dirigirnos al apartado *View License Status*, donde aparecerán todas las licencias activadas. En la siguiente Figura 73 aparece la licencia cargada anteriormente.

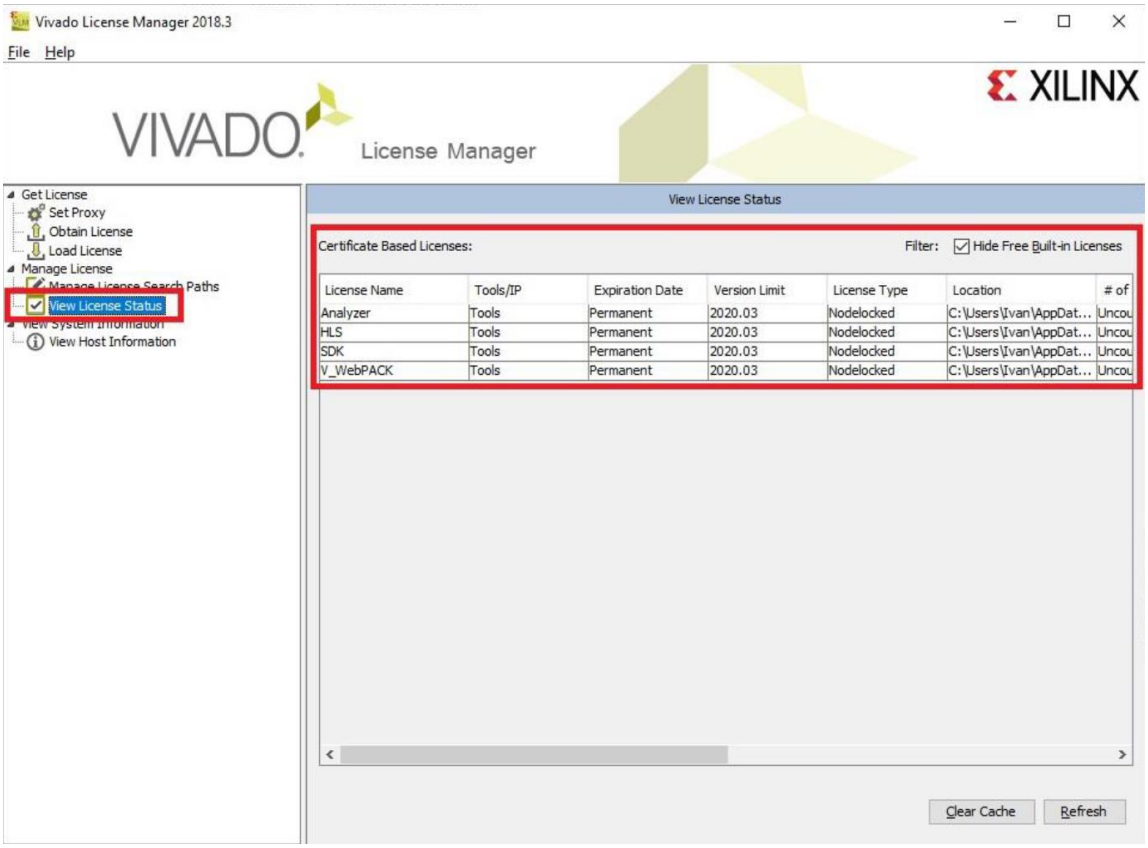


Figura 73: Descarga Vivado – Licencia - Parte 8

Con todo este proceso completado, ya se tienen configuradas la licencias. El último paso restante para tener completada la configuración de la instalación de Vivado es la configuración de los ficheros de la placa Nexys 4 DDR de Digilent, que es la utilizada para este proyecto. Esto se realiza para que Vivado pueda reconocer la placa.

Para esta configuración, es necesario descargar los ficheros de la placa, accediendo al repositorio de la placa de Digilent a través del siguiente enlace de descarga: https://github.com/Digilent/vivado-boards/archive/master.zip.

Una vez descargado el archivo .zip, es imprescindible descomprimir el archivo y abrir los ficheros. Seguidamente se debe acceder a la carpeta *vivado-board-master\new\boards-files*; y se seleccionan todas las carpetas para posteriormente copiarlas. Una vez se han copiado, hay que pegarlas en la siguiente carpeta: *C:\Xilinx\Vivado\2018.3\data\boards\board_files*. En la Figura 74 se puede comprobar este proceso.

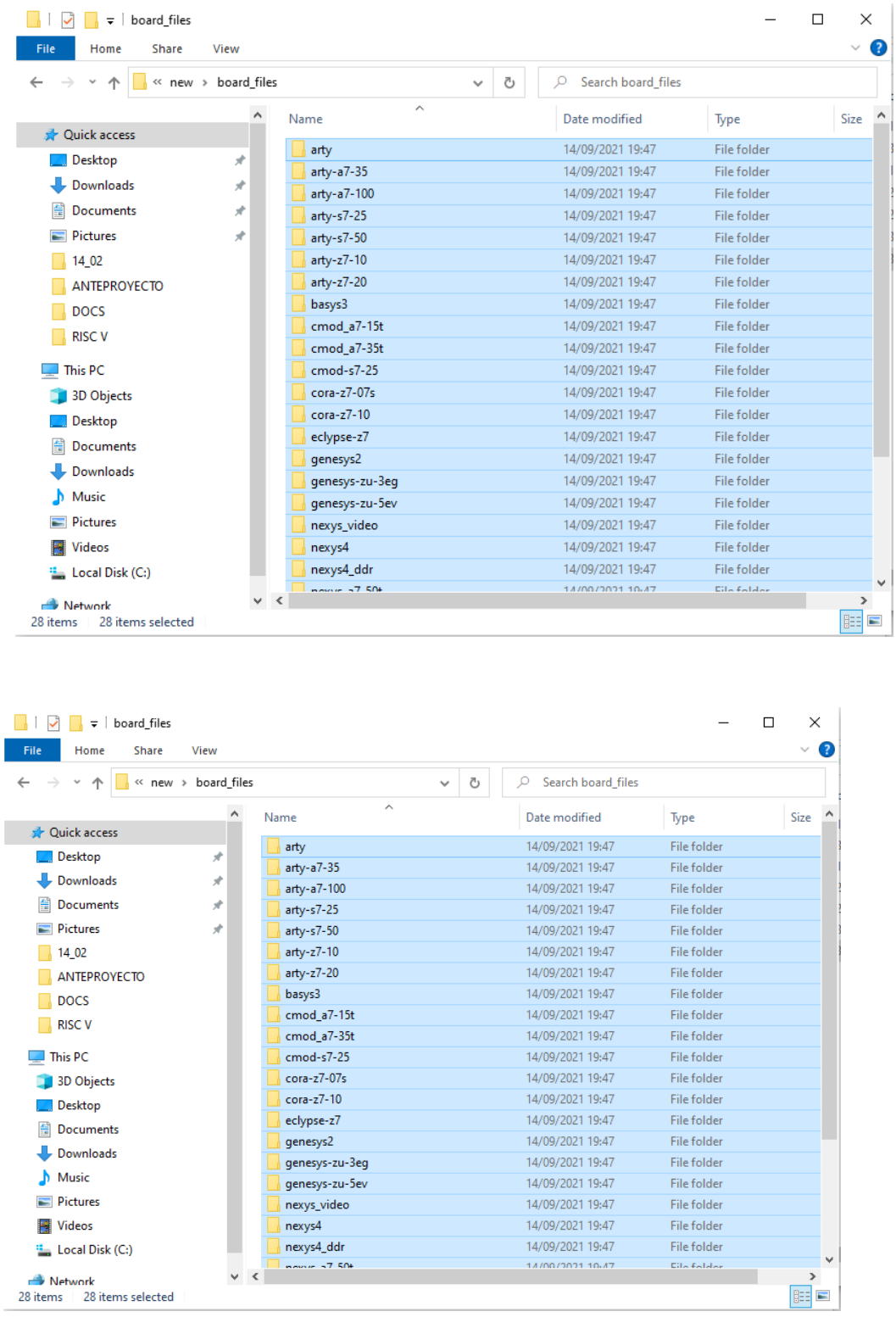


Figura 74: Descarga Vivado – Licencia - Parte 9

ANEXO 2: VIVADO BASICS

En este anexo se exponen las diferentes partes que componen Vivado, en cuanto al análisis del diseño, especialmente aquellas que son más importantes para la realización de este proyecto.

La vista principal de Vivado tiene diferentes secciones tal y como se aprecia en la siguiente Figura 75.

1. *Flow Navigator* – Navegador principal
2. *Sources* – Fuentes
3. *Properties* – Propiedades
4. *Workspace* – Espacio de trabajo
5. *Results* – Resultados

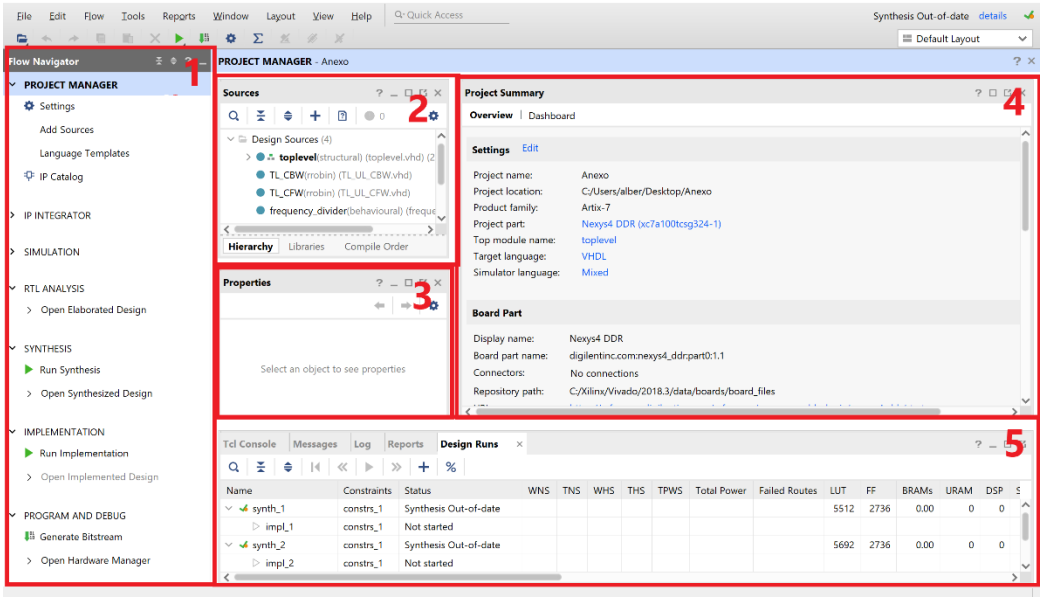


Figura 75: Partes de Vivado

1. *Flow Navigator*

En la parte de la izquierda de Vivado, se puede observar la sección llamada *Flow Navigator*. Esta parte de Vivado permite controlar la mayoría de las tareas del proceso del diseño, como la configuración, síntesis, implementación o generar el *bitsream* Figura 76. A continuación, se numeran todas estas herramientas y se explican las más relevantes.

- *Project manager*

En esta sección se puede acceder a la configuración del proyecto. Es un acceso rápido para modificar los ajustes del proyecto (*settings*), añadir nuevos ficheros (*add sources*), plantillas para la programación de los circuitos (*Language Templates*) y un catálogo IP.

- *IP Integrator*

Herramientas para crear bloques del diseño (*Create Block Design*).

- *Simulation*

Permite verificar las salidas antes de programar el dispositivo (*Run Simulation*).

- *RTL Analysis*

Permite la interpretación del código, es decir, elabora el circuito con las estructuras hardware reales.

- *Synthesis*

En la *Synthesis* se halla la sección más importante para este proyecto, por tanto, se explica con un poco más de detalle más adelante. Aquí se sintetiza el circuito pudiendo establecerse diferentes ajustes de síntesis, además de poder obtener reportes post-síntesis. Los apartados que incluye son:

- *Constraints Wizard*: el asistente de restricciones de tiempo identifica qué restricciones de tiempo faltan por añadir en la síntesis o implementación del diseño.

- *Edit timing Contraints*: utilizado para editar las restricciones de tiempo del diseño.

- *Set Up Debug*: este proceso se realiza para configurar la detección de las salidas en el circuito cuando se esté en la fase de implementación en la placa, y para la depuración del sistema diseñado.

- *Report Timing Summary*: esta es una de las partes más importantes en cuanto al cometido de este trabajo, ya que desde aquí se interpretarán todos los tiempos relativos al camino del sistema, pudiendo extraer de ello la información necesaria para la mejora del procesador. Se hablará extensamente de ello en el ANEXO 4: REPORTE DE TIEMPO DE VIVADO.

- *Report Clock Networks*: este reporte provee una vista de todos los relojes en el sistema, en forma de “árbol”. En este caso, solo se trabaja con un reloj.

- *Report clock interaction*: muestra el resultado de la interacción entre los relojes existentes en un sistema.

- *Report Methodology*: es un reporte que comprueba el diseño del circuito sintetizado, mostrando las advertencias y errores que tiene el diseño.
- *Report DRC* (*Design Rules Check*): comprueba el diseño y reporta incidencias comunes.
- *Report Noise*: calcula el SSN, *Simultaneous Switching Noise* (en español: Ruido de conmutación simultáneo) de la placa con la que se está trabajando.
- *Report Utilization*: este reporte ayuda a analizar la utilización del diseño con diferentes recursos, a nivel jerárquico, *Pblocks* definidos por el usuario o SLR (*Super Logic Region*)
- *Report Power*: reporte para conocer el consumo del sistema.
- *Schematic*: obtiene un esquema del diseño.

- *Implementation*

Permite el acceso a los ajustes y herramientas para la implementación del circuito.

- *Program and debug*

Este último apartado genera el *Bitstream*, para volcar el programa en la FPGA.

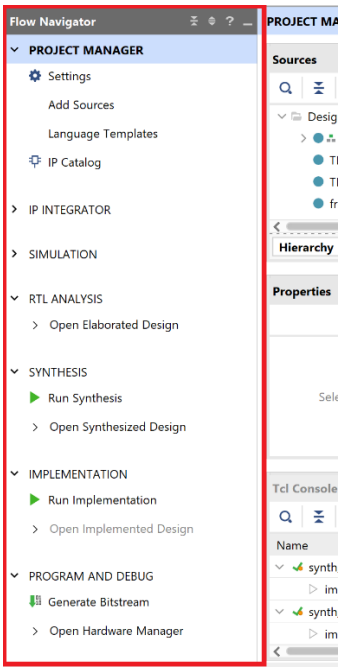


Figura 76: Vivado - Flow Navigator

2. Sources

En esta sección, se establece el proyecto de forma jerárquica donde se puede ver su estructura formada por los archivos que lo componen.

La carpeta *Design Sources* contienen los archivos HDL que componen el sistema. Las restricciones están guardadas en la carpeta *Constraints* y los archivos de simulación en la carpeta *Simulation Sources*. Haciendo doble clic en un archivo, conseguimos abrir su código en el *Workspace*. Se pueden añadir nuevos archivos al diseño haciendo clic derecho en la carpeta y clic en *Add Sources* o directamente en el botón *Add Sources*. También se puede acceder a las librerías creadas del proyecto en la pestaña *Libraries* (Figura 77).

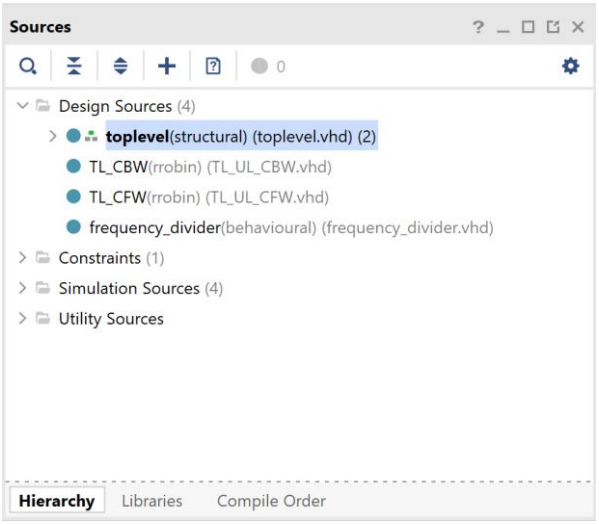


Figura 77: Vivado - Sources

3. Properties

En *Properties*, aparecen las propiedades de los archivos seleccionados en el panel *Sources*. Aunque no es muy importante para este proyecto, puede ser utilizado para ver y editar las propiedades de un archivo (Figura 78).

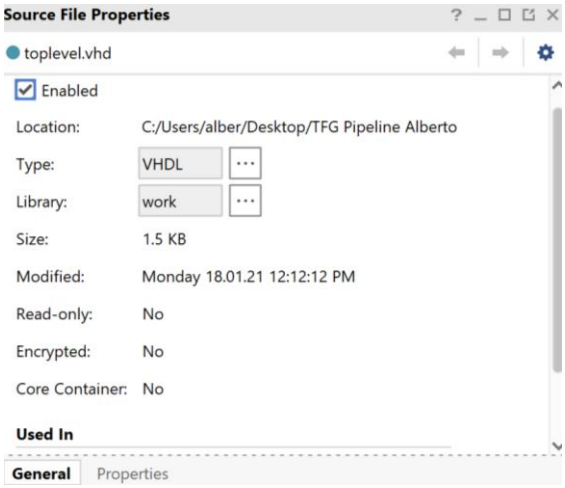


Figura 78: Vivado – Properties

4. *Workspace*

Este apartado es el más importante del contenido de *Project Manager* en Vivado. Workspace nos permite abrir los reportes y editar los archivos HDL/restricciones. En el momento inicial en este panel se abre el *Project Summary* (Figura 79).

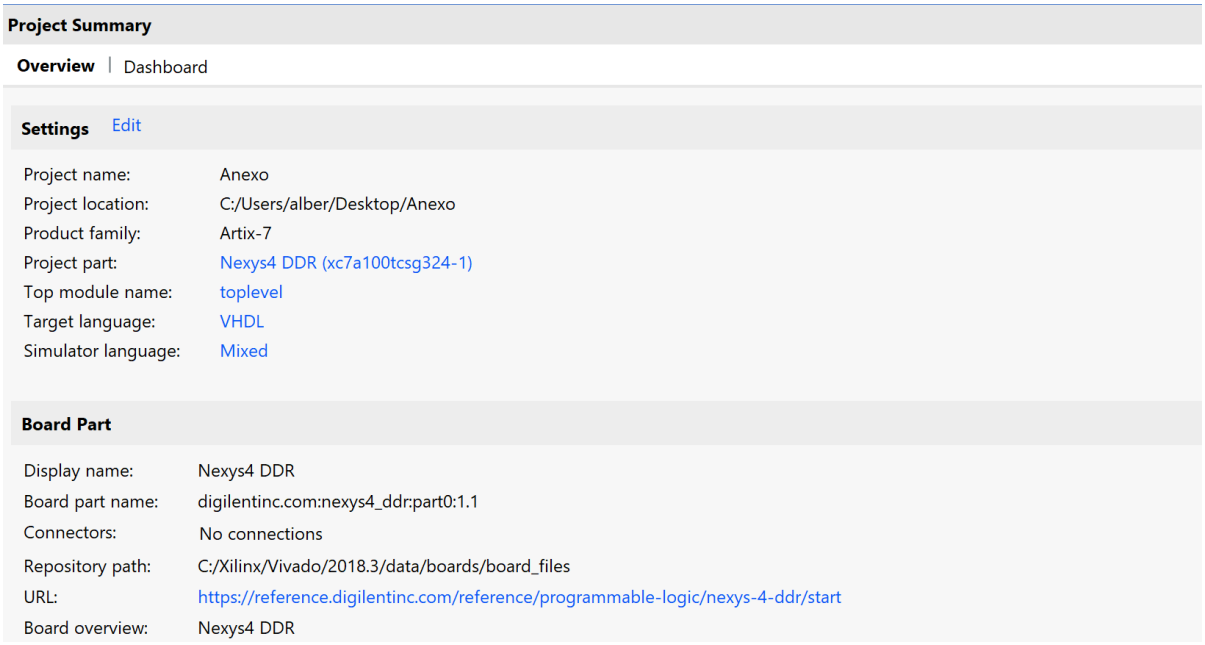


Figura 79: Vivado - Workspace

5. *Results*

Esta parte del programa tiene múltiples herramientas bastantes útiles, cómo *TCL Console*, *Messages*, *Reports* o *Log* (Figura 80).

- *TCL Console*: es una consola para ejecutar comandos directamente sin tener que utilizar la interfaz de Vivado.
- *Messages*: esta herramienta informa de todos los mensajes de error, avisos u otro tipo de información importante.
- *Reports*: en este apartado se pueden comprobar los diferentes reportes que ofrece el programa: reportes de tiempo, de potencia, de utilización, etcétera.
- *Log*: muestra los resultados de la última síntesis, implementación y simulación. No suele ser necesario, ya que se puede obtener esta información más detallada en los reportes y mensajes.

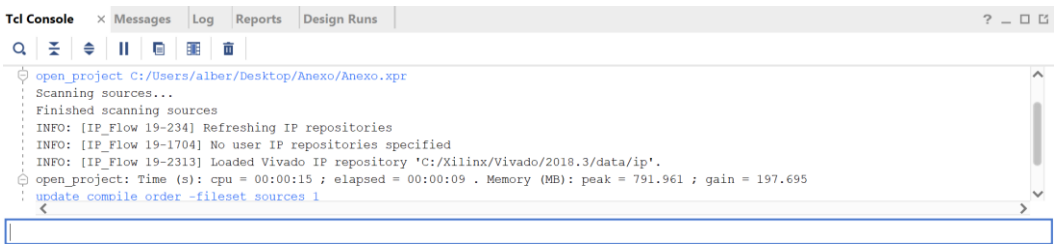


Figura 80: Vivado - Results

ANEXO 3: MANUAL DE USUARIO DE VIVADO

En este apartado se indica como crear un nuevo proyecto en Vivado. Se parte de un proyecto vacío en el cual se le añadirán los ficheros del sistema ya implementado para la posterior mejora del Pipeline.

El primer paso será ejecutar Vivado en el equipo (Figura 81). Una vez abierta la ventana del programa, se creará un nuevo proyecto haciendo clic en *Create Project*. (Figura 82) Una vez hecho esto, se abre una ventana nueva, para la configuración del proyecto. Se debe pulsar *Next* para el siguiente paso (Figura 83).

Figura 81: Miniatura de Vivado (Escritorio)

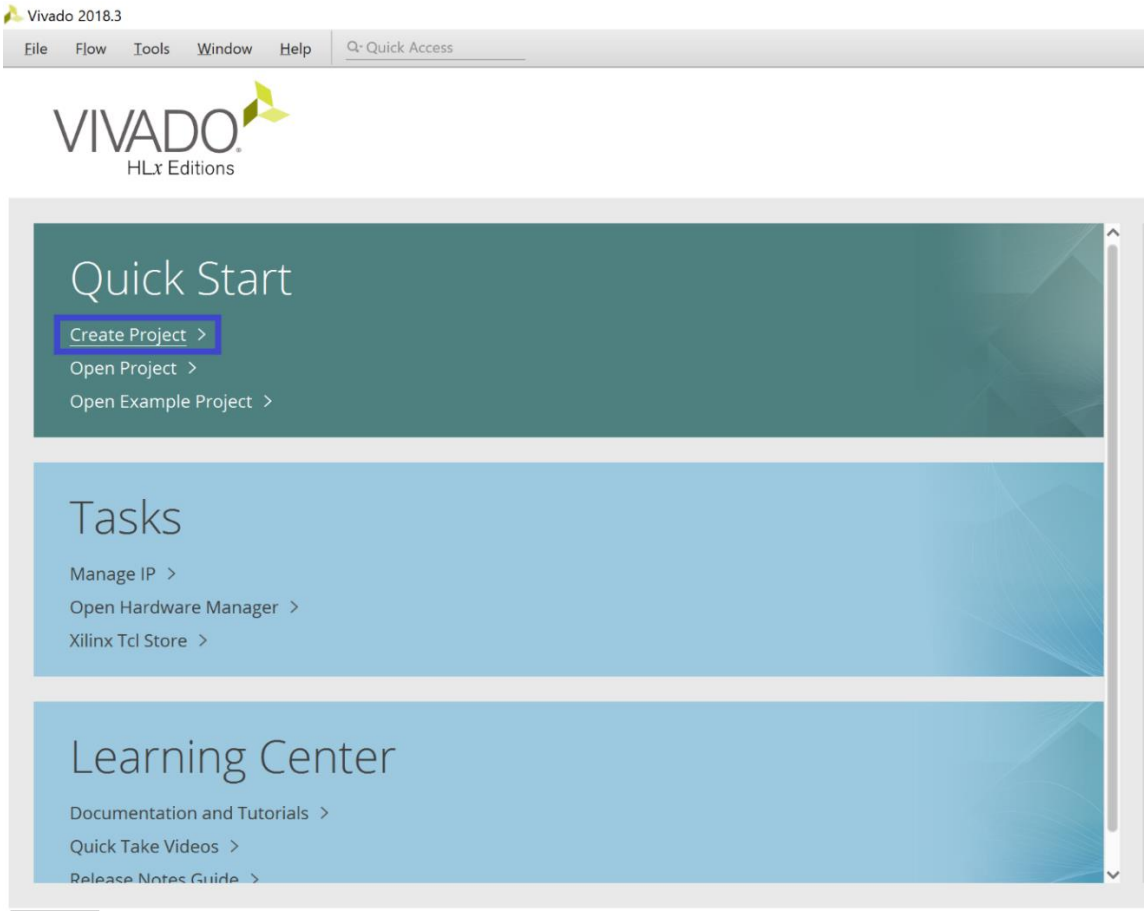


Figura 82: Vivado - Pantalla de inicio – Creación de un nuevo proyecto – Parte 1

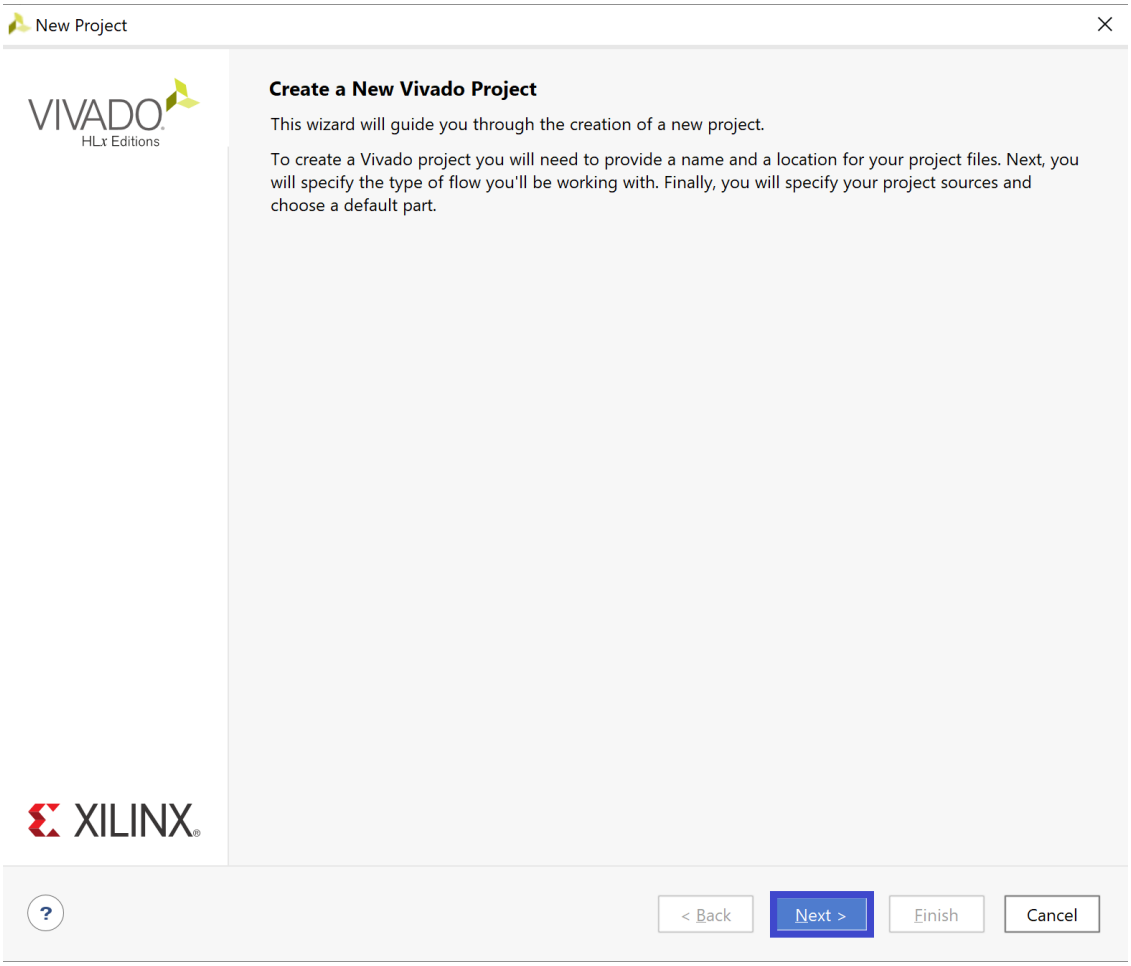


Figura 83: Vivado - Pantalla de inicio – Creación de un nuevo proyecto – Parte 2

A continuación, se introducirá un nombre para el proyecto. En este caso se ha utilizado el nombre "Anexo" para la realización de este manual, pero en casos futuros, el nombre es de libre elección del usuario. También se escogerá la ubicación de nuestro equipo donde se desea guardar el proyecto (Figura 84).

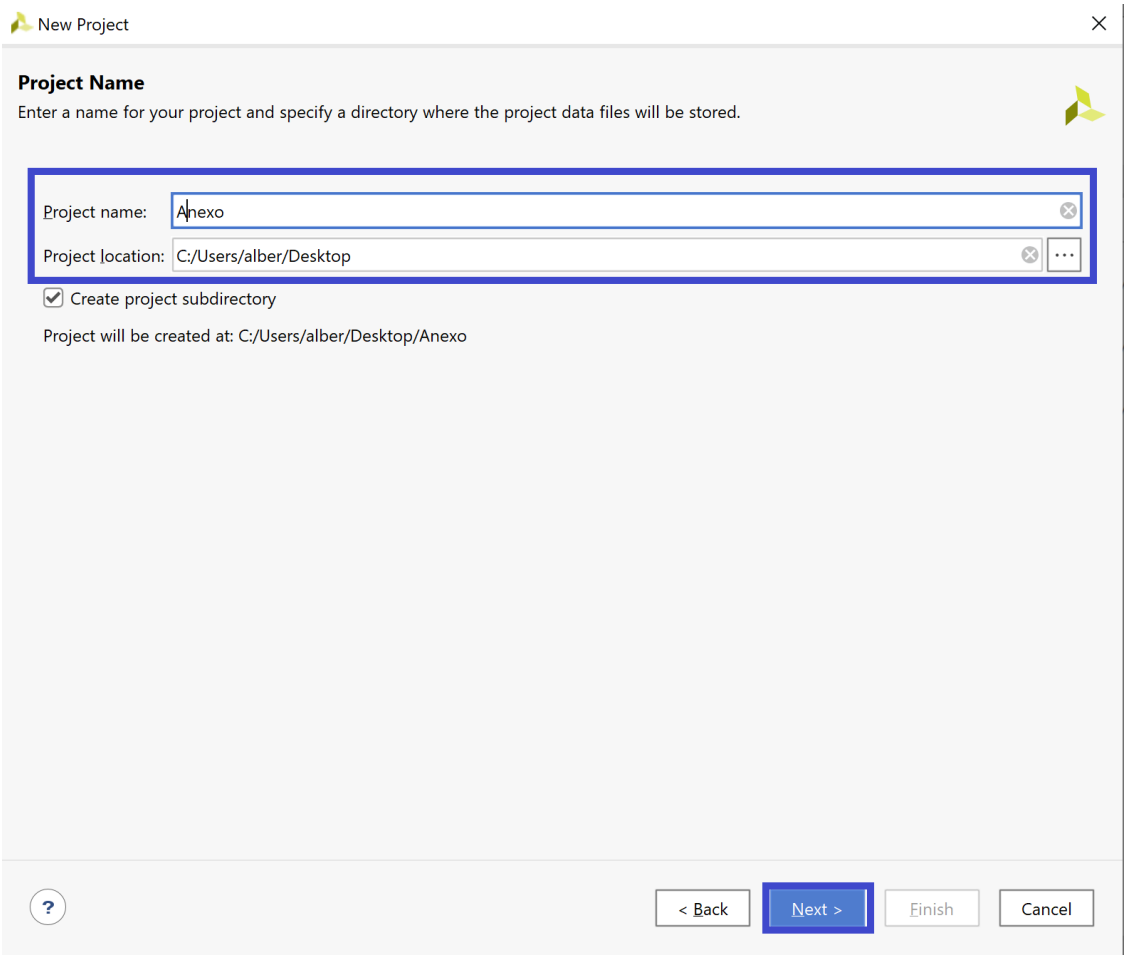


Figura 84: Vivado - Nombre del nuevo proyecto

En la siguiente sección se define el tipo de proyecto a crear. En el caso de estudio que nos ocupa se elige RTL Project, y además se selecciona el recuadro *Do not specify sources at this time* (en español, no especificar fuentes en este apartado), ya que éstas se incluirán más tarde (Figura 85). Cuando se elige RTL Project, se pueden añadir ficheros fuente, analizar el diseño RTL, sintetizarlo e implementarlo y obtener diferentes análisis.

Seguidamente, se elige la placa sobre la que se va a trabajar en el diseño del procesador. Para ello se selecciona el apartado *Boards*, y a continuación se introduce en la búsqueda la placa deseada. Para este trabajo, se ha elegido la placa *Nexys4 DDR de Digilent*. Por último, seleccionar *Next* (Figura 86).

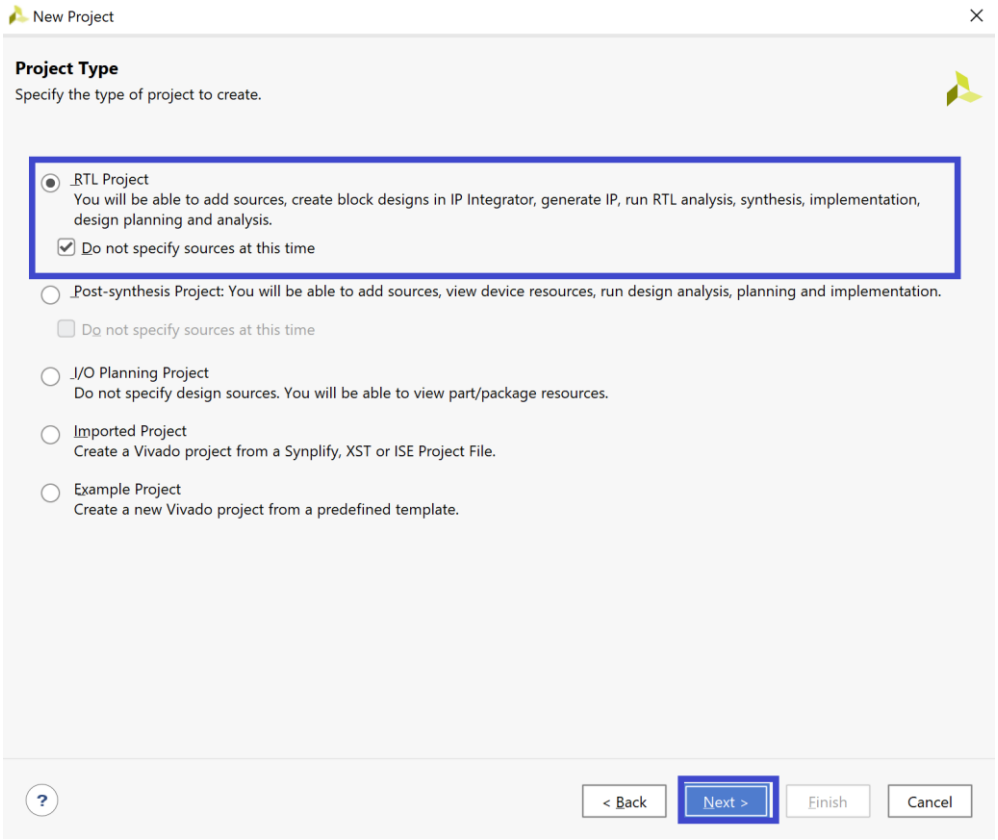


Figura 85: Vivado - Tipo de proyecto

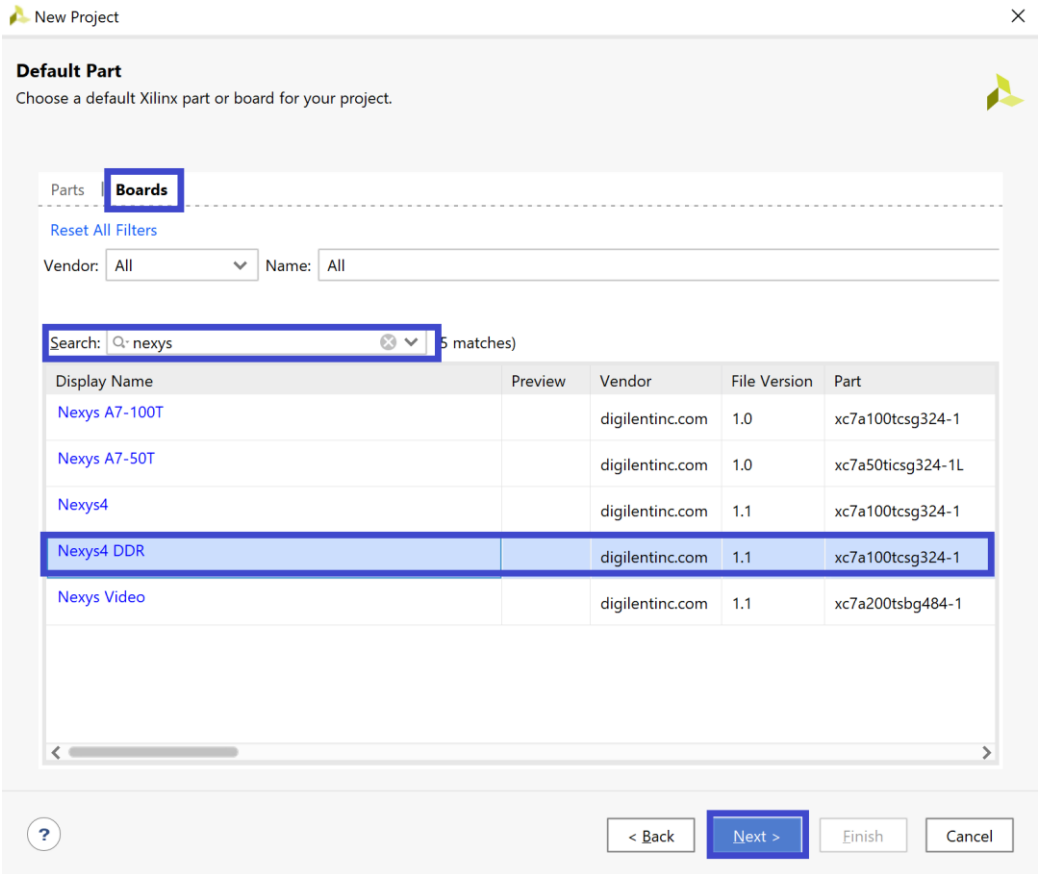


Figura 86: Vivado - Elección de la placa

Finalmente, se llega a la última sección donde aparece un resumen de la configuración que se ha elegido para el proyecto. Para cerrar esta ventana de diálogo se debe seleccionar *Finish* (Figura 87).

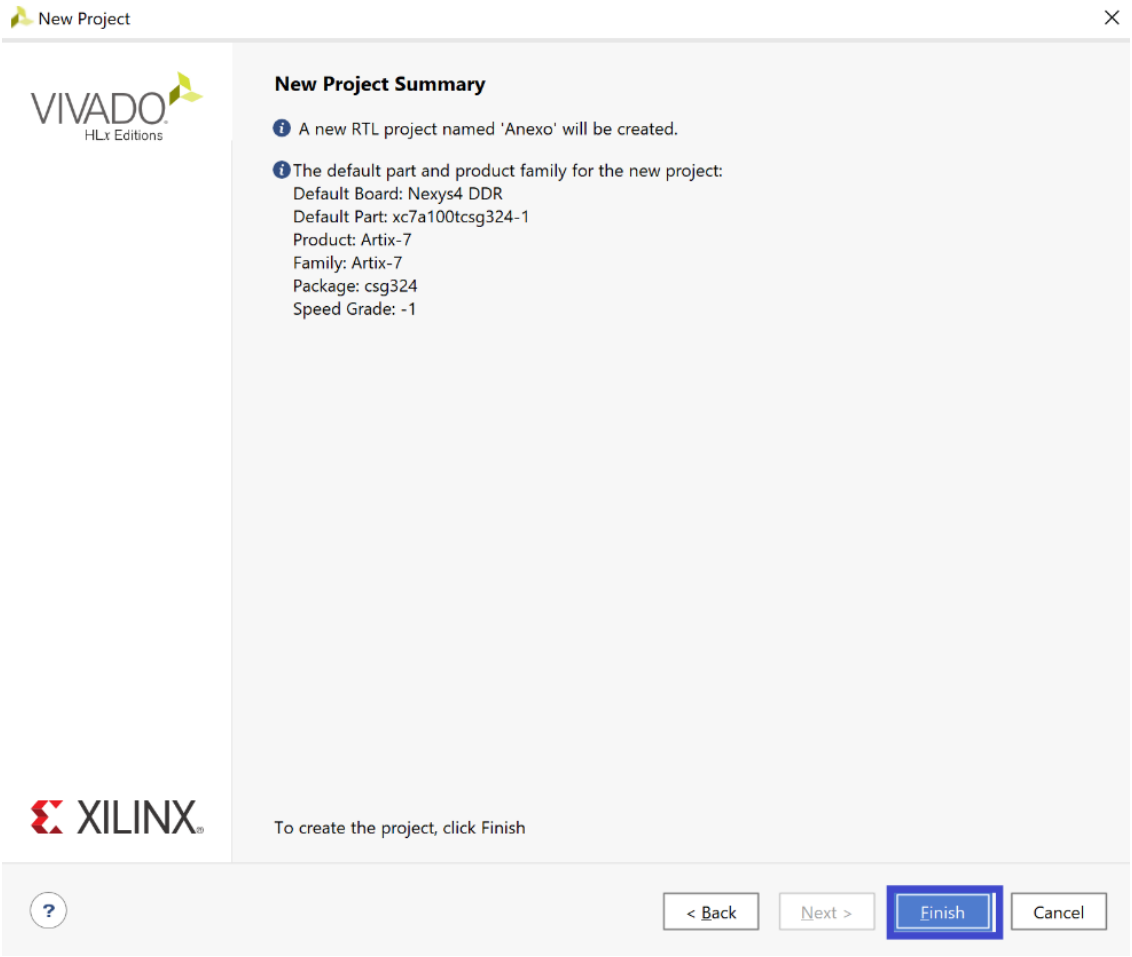


Figura 87: Vivado - Resumen de la configuración del proyecto

Una vez creado el proyecto, se abre automáticamente el programa. Lo primero que se necesita hacer es configurar los ajustes principales del proyecto. Seleccionar *Settings* ubicado en la sección del *Flow Navigator* (Figura 88).

Lo único que se debe modificar en los ajustes, se encuentra en el apartado de ajustes generales. En el apartado *Target Language* se debe de modificar el lenguaje a usar, VHDL. Además, se tiene que cambiar el nombre de la librería por defecto, *Default library*. El nombre que se le va a dar es "*work*" (Figura 89).

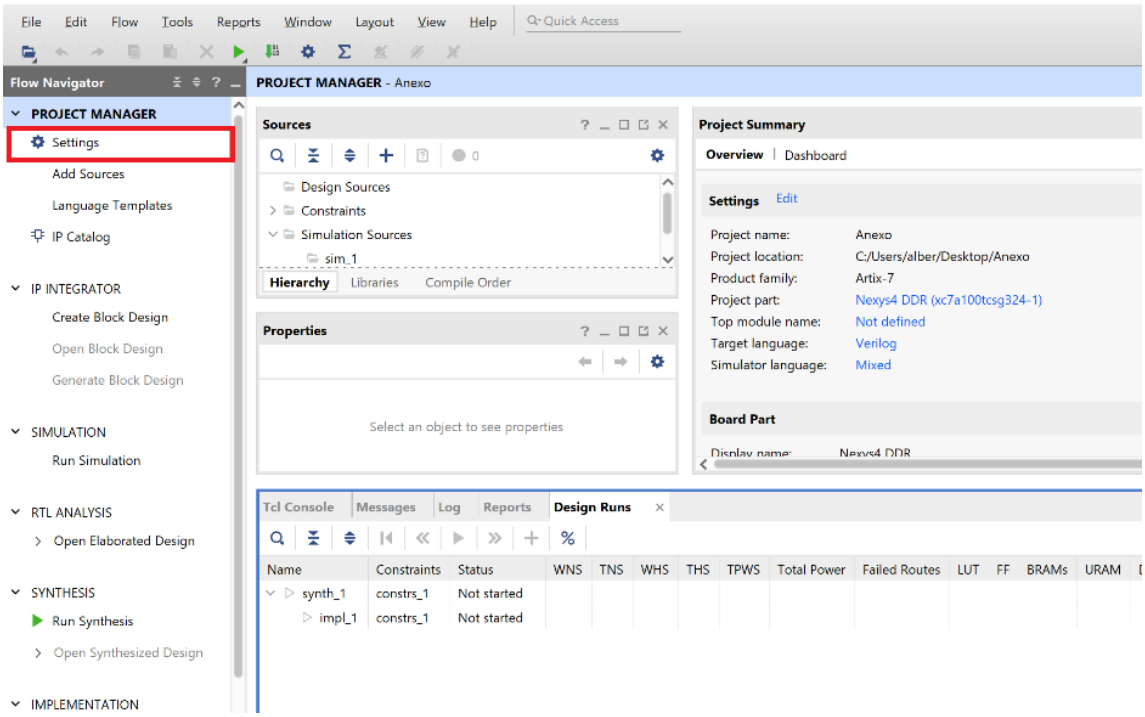


Figura 88: Vivado – Settings – Parte 1

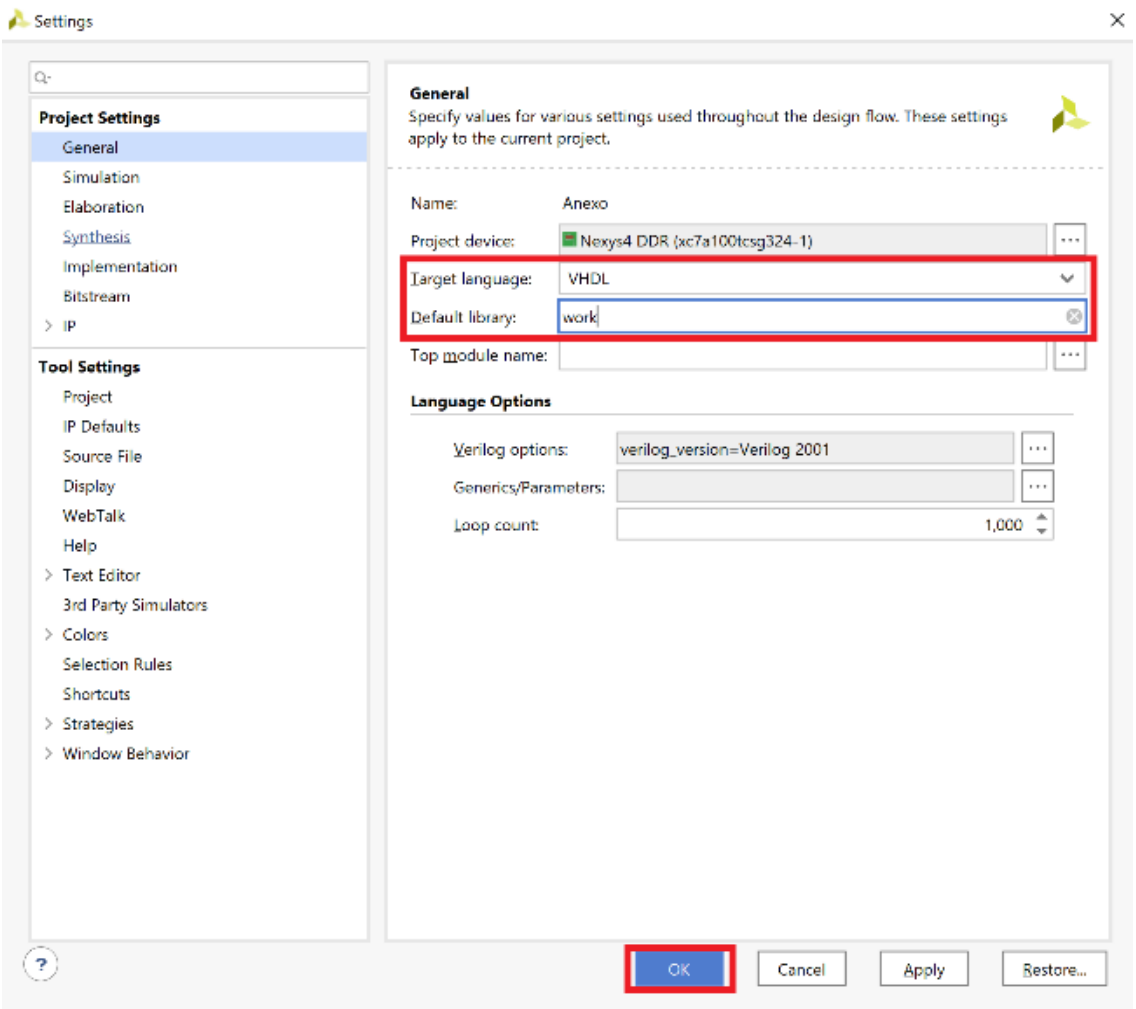


Figura 89: Vivado - Setting- Parte 2

Ahora es el momento de incluir los ficheros del proyecto, los cuales han sido descargados con anterioridad. Para ello es necesario localizar la sección *Sources*, donde se tiene que hacer clic en el icono *SIGNO* +, tal y como se muestra en la Figura 90:

Nota: Para la descarga del código es necesario pedir el permiso al SRG, en concreto a Iván Gamino del Río y a Agustín Martínez Hellín.

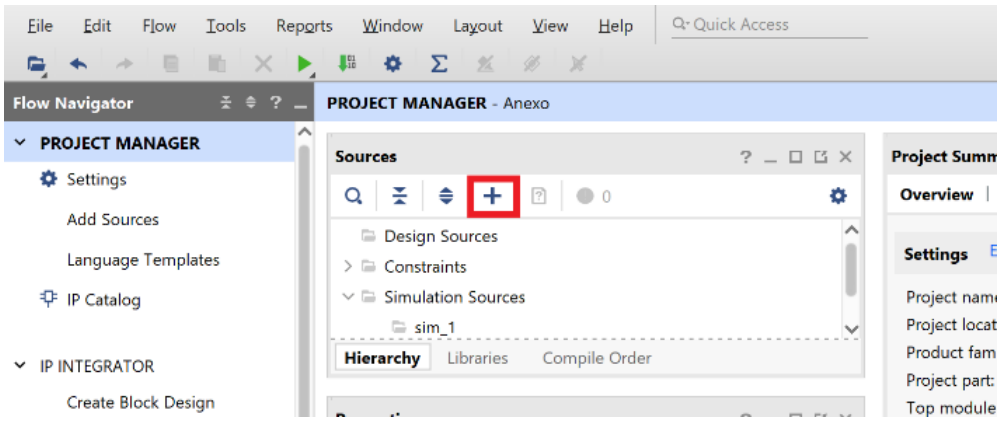


Figura 90: Vivado - Añadir ficheros al proyecto – Parte 1

Una vez pulsado el icono, aparece una nueva ventana con tres tipos de archivos fuentes: de restricciones, de diseño y de simulación. El proyecto que se está desarrollando se compone únicamente de ficheros de diseño y de simulación. En este caso se deja la opción marcada de archivos de diseño *Add or create design sources* (Figura 91).

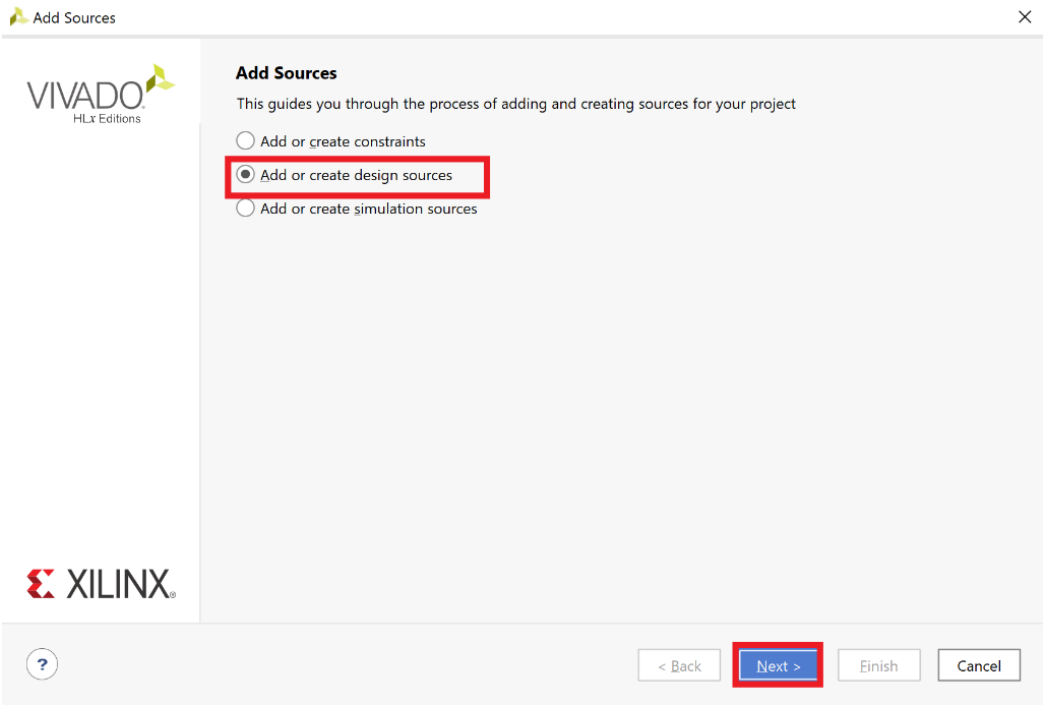


Figura 91: Vivado - Añadir ficheros al proyecto – Parte 2

A continuación, en la siguiente ventana se pueden añadir los ficheros descargados incluyendo el directorio del equipo donde se encuentren. Para ello, se debe seleccionar añadir directorio, *Add Directories*, así como muestra la siguiente Figura 92.

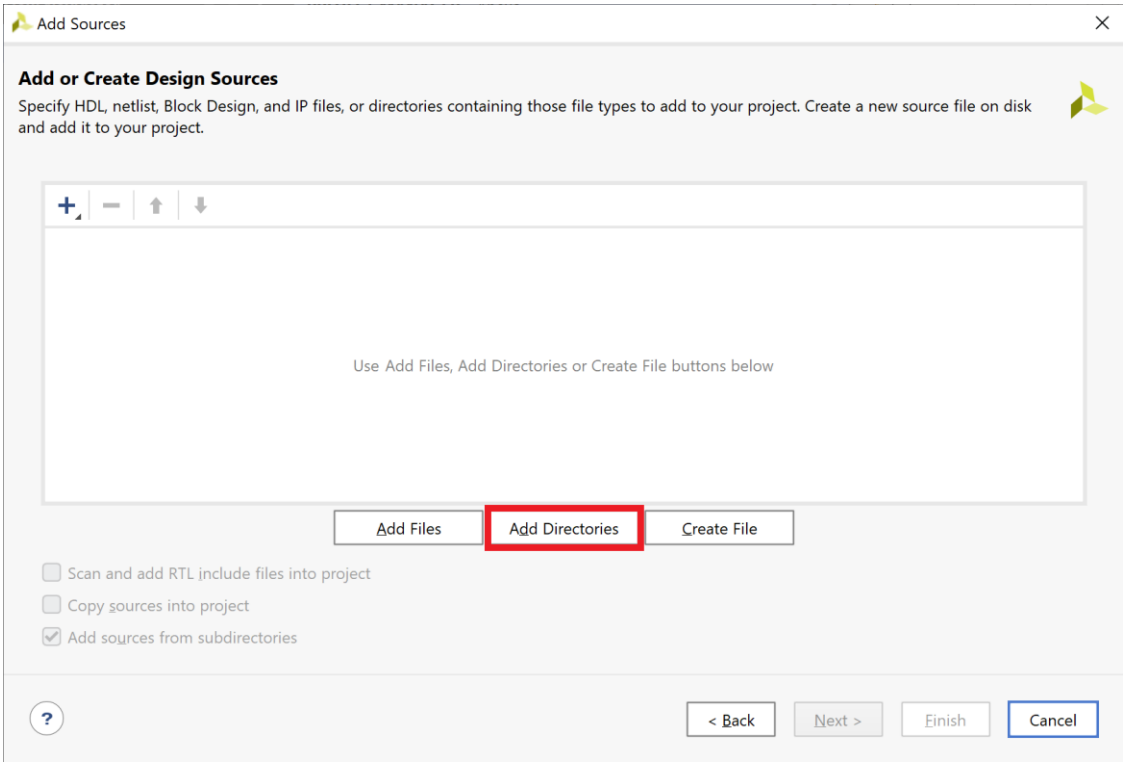


Figura 92: Vivado - Añadir ficheros al proyecto – Parte 3

En este momento, se busca el directorio donde se encuentra el proyecto en la ventana emergente, se selecciona la carpeta *design*, y se hace clic en *Select*. Una vez hecho esto, el archivo debería estar incluido, tal y como se aprecia en la Figura 93. Finalmente, se hace clic en *Finish* (Figura 94).

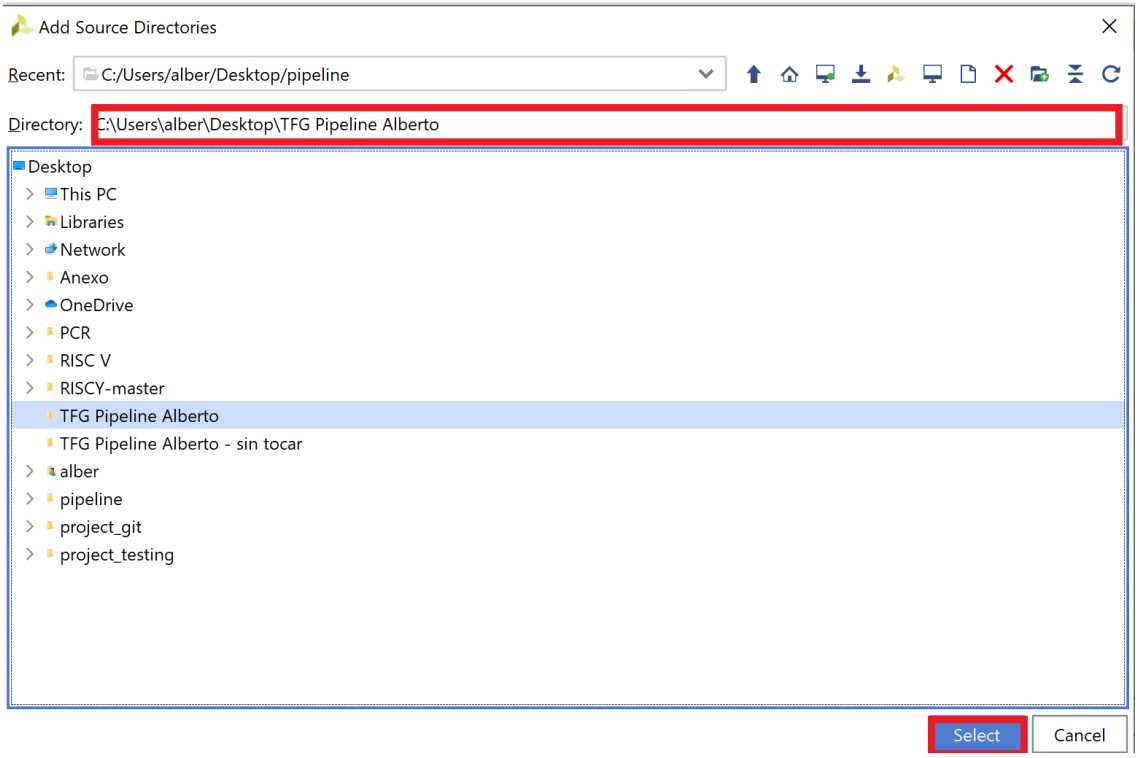


Figura 93: Vivado - Añadir ficheros al proyecto – Parte 4

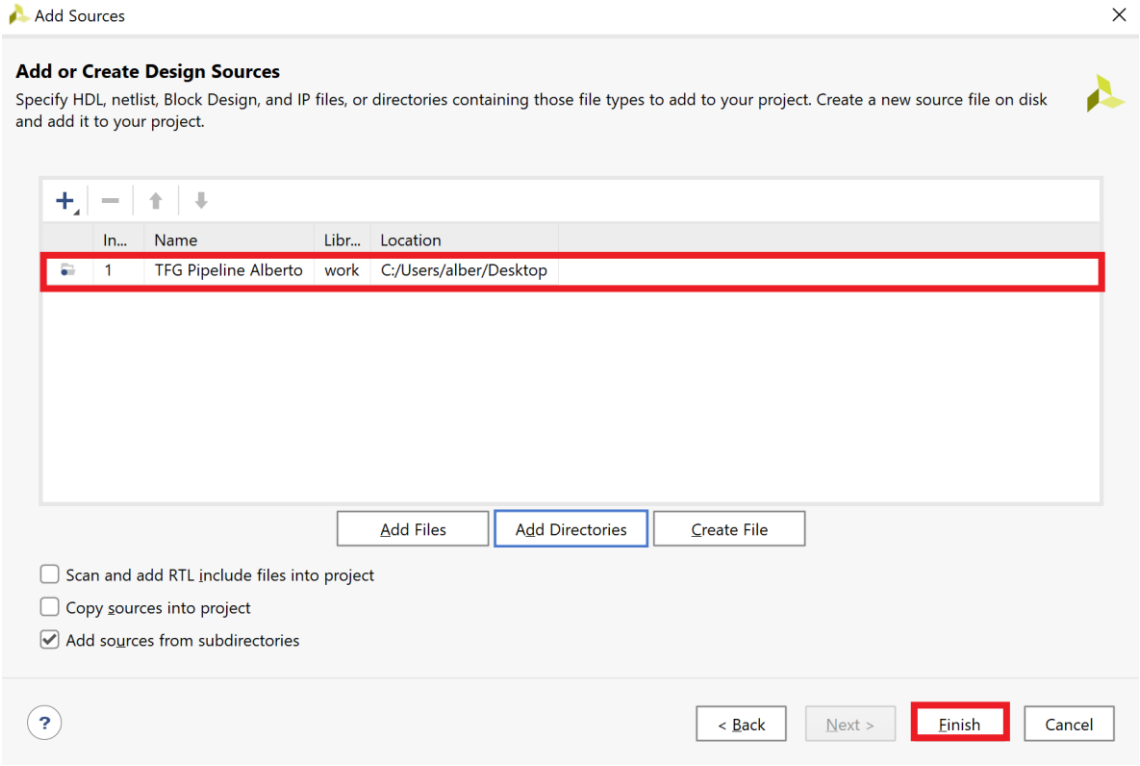


Figura 94: Vivado - Añadir ficheros al proyecto – Parte 5

En la siguiente figura podemos ver como ha quedado la ventana principal de Vivado una vez que se han incluido los ficheros del proyecto donde se va a trabajar. Se deben configurar algunas partes del diseño, por lo que, para ello, en la parte de la izquierda, en el *Flow Navigator*, se hace clic en *Open Elaborated Design* en la sección *RTL Analysis* (Figura 95). Seguidamente en la ventana que aparece, se hace clic en OK.

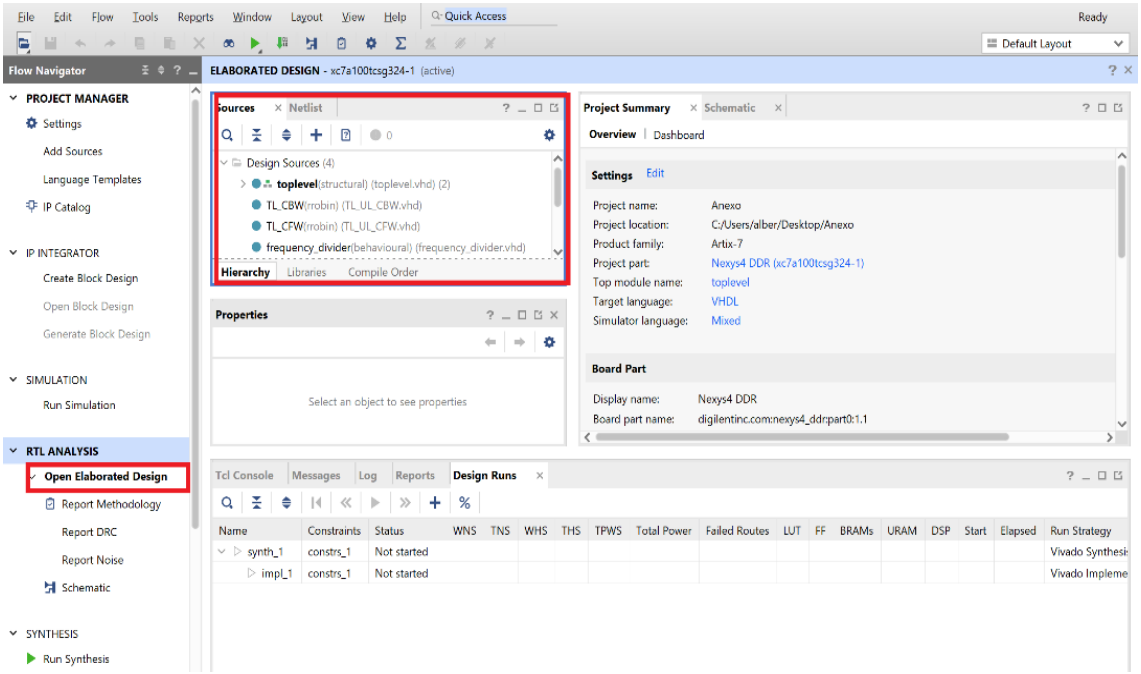


Figura 95: Vivado - RTL Analisys - Open Elaborated Design

Una vez realizado el paso anterior, emerge una vista diferente en Vivado, donde aparece una nueva sección llamada *Elaborated Design*. Ahora se deben configurar los puertos de la placa que conciernen a este proyecto. En este caso, en la parte de la barra de herramientas, se puede buscar la sección *I/O Ports*, como se muestra en la Figura 96.

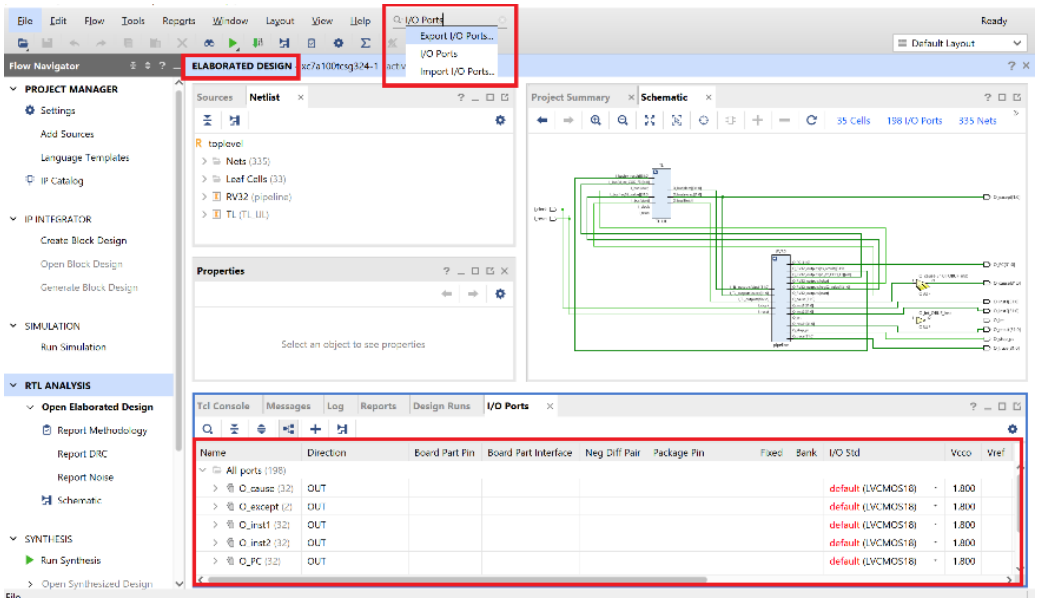


Figura 96: Vivado - Edición de los puertos de la placa – Parte 1

Las entradas y salidas que se deben modificar son las mostradas en la siguiente Figura 97 (*Scalar ports*). Se ha accedido al datasheet de la placa [8], para comprobar qué pines son los que se deben configurar. Por ejemplo, el pin del reloj (*clock*) tiene que ser el pin del reloj de la placa, E3, el cual funciona a 100 MHz. El pin de *reset* se ha adjudicado al pulsador N17. En cuanto las salidas ´stop´ e ´ínt´ se han adjudicado a los pines H17 y K15, respectivamente. Además, existen otros puertos que son adjudicados por Vivado automáticamente, estos son fijados para que se adjudiquen correctamente a la hora de implementarlos en placa. Para ello se debe dejar marcada la casilla *Fixed* de cada uno de los puertos que aparecen.

Una vez establecidos los pines de la placa, se debe guardar el proyecto, ya que, en este momento, el programa notifica los cambios que van a aparecer con un asterisco en *Elaborated Design*.

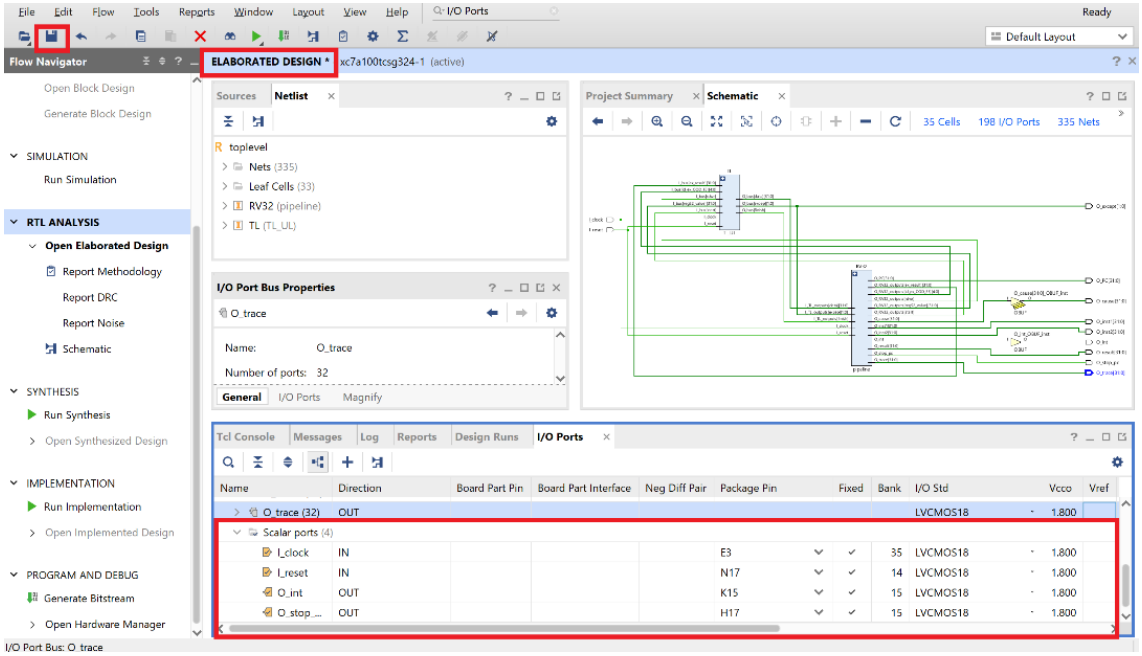


Figura 97: Vivado - Edición de los puertos de la placa – Guardar

Inmediatamente después, emerge una ventana preguntando por el nombre del fichero que guardará estos cambios. Este fichero es el fichero de restricciones, en el que se almacena la información con los puertos de entrada/salida entre la placa y el diseño. El nombre que se le pone es indiferente, pero en este caso, se le ha llamado *constraints*. A continuación, pulsar el botón OK (Figura 98).

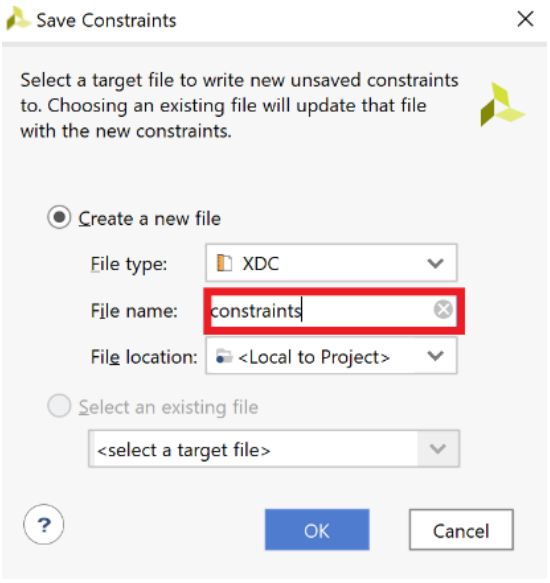


Figura 98: Vivado - Constraints

Después de este proceso, en Vivado se muestra este fichero recién creado de restricciones, donde se pueden ver los pines asignados en la placa. Ya está todo listo para continuar con la siguiente parte, la síntesis del circuito. En la parte del *Flow Navigator*, pulsar *Run Synthesis*, como se indica en la Figura 99.

Inmediatamente después aparece una ventana como la de la Figura 100. Entonces pulsar OK.

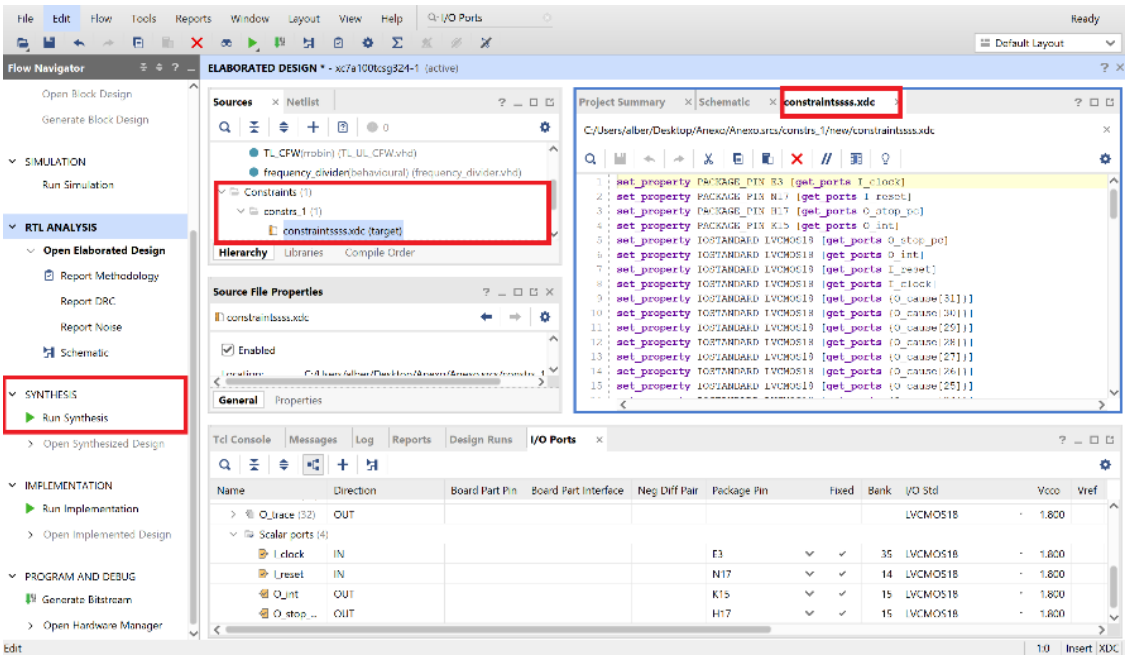


Figura 99: Vivado - Constraints - Síntesis

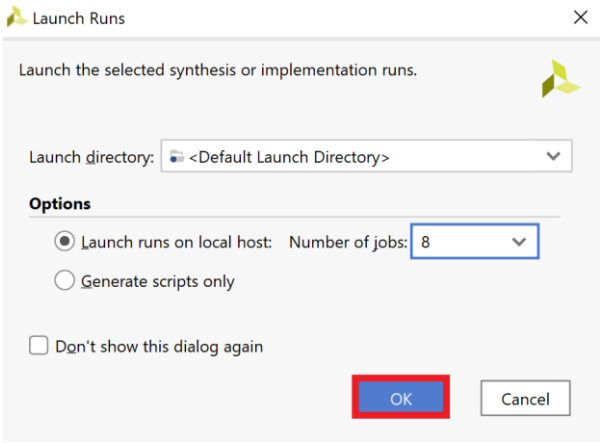


Figura 100: Vivado - Síntesis - Ejecución

En estos momentos está sintetizándose el circuito, este proceso puede durar unos minutos hasta completarse. Una vez completada la síntesis, nos preguntará si queremos realizar la implementación del circuito, abrir el diseño sintetizado o ver los reportes. Para el cometido de este trabajo y para ahorrar tiempo de procesamiento, se trabajará sobre el circuito sintetizado, ya que es bastante más rápido a la hora de validar los cambios que se vayan realizando, aunque una vez hecha la mejora, se implementará el circuito y se harán las pruebas pertinentes en placa. Para realizar las modificaciones, se selecciona *Open Synthesized Design* y se pulsa OK (Figura 101). Cuando las modificaciones estén realizadas se seleccionará *Run Implementation*.

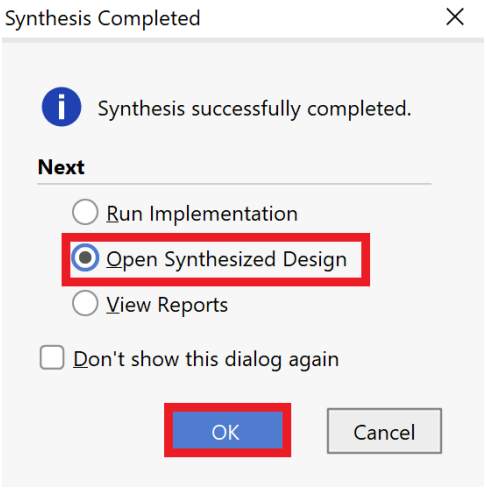


Figura 101: Vivado - Síntesis - Abrir diseño sintetizado

Después de este paso, se introducen las restricciones de tiempo. Para ello, en el apartado de *Flow Navigator*, en la sección *Synthesis* se debe hacer clic en *Edit Timing Contrains* para editar el reloj que se usará en la simulación del procesador. En el apartado indicado en la figura como *Timing Constraints*, se debe hacer doble clic donde aparece *Double click to create a Create Clock constraint* (Figura 102).

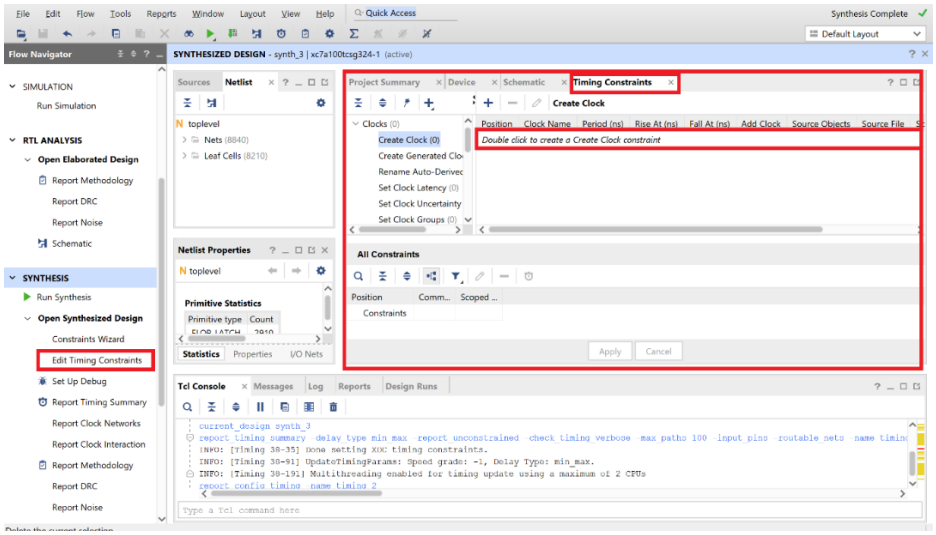


Figura 102: Vivado - Síntesis - Editar restricciones de tiempo – Parte 1

Seguidamente aparecerá una nueva ventana en la que es necesario especificar un nombre para el reloj y definir su periodo. La placa que se utilizará para la implementación del procesador es una Nexys 4 DDR, por lo que, según sus especificaciones, la frecuencia de reloj máxima es de 100 MHz, aunque es este proyecto se utilizará una frecuencia de 50 MHz para el procesador, por lo que el periodo de la señal de reloj establecido inicialmente será de 20 ns. Como se muestra en la siguiente figura, el nombre de la señal de reloj ha sido, a modo de ejemplo, *clock_anexo*, con un periodo de 20 ns. Una vez rellenados los campos, se debe hacer clic en OK para guardar los cambios (Figura 103).

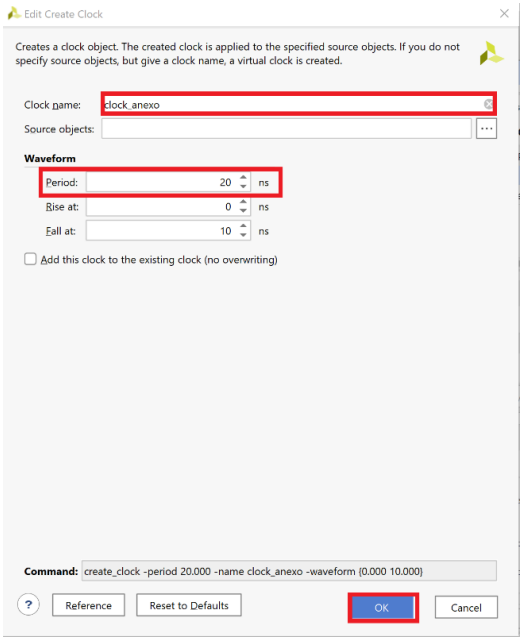


Figura 103: Vivado - Síntesis - Editar restricciones de tiempo – Parte 2

Una vez realizados todos los pasos explicados anteriormente, ya se ha configurado el proyecto en Vivado.

ANEXO 4: REPORTE DE TIEMPO DE VIVADO [6]

En este anexo, se explica cómo obtener los reportes de tiempos del sistema, así como la explicación de cada término, ahondando más en aquellos importantes para la realización de la mejora del Pipeline del procesador en RISC-V.

Existen varias maneras de obtener el reporte de tiempos. Una de las formas es directamente desde la sección *Flow Navigation*, en el apartado de *Synthesis*, haciendo clic en *Report Timing summary*. La otra es desde la barra de herramientas de Vivado, en la pestaña *Reports* → *Timing* → *Report timing Summary*. En la Figura 104 se aprecia el reporte que se obtiene desde la sección *Flow Navigator*.

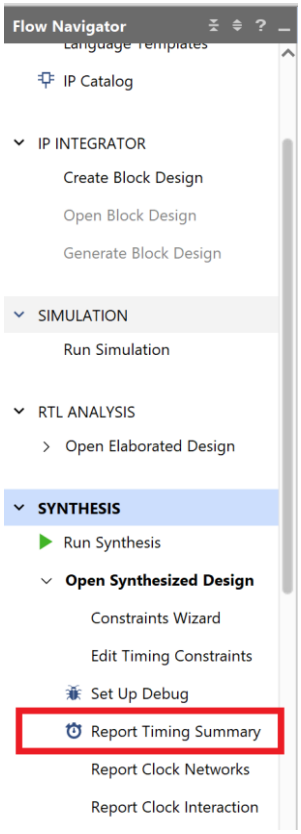


Figura 104: Vivado - Report Timing Summary – Parte 1

A continuación, se visualiza una ventana como la mostrada en la siguiente Figura 105 donde puede configurarse el modo en el que se quiere realizar el reporte de tiempo. Se debe establecer un nombre para el reporte que se está obteniendo (timing_1, por defecto). Existen tres pestañas diferentes en esta ventana, que son: *Options*, *Advanced* y *Timer Settings*. Al pie de esta ventana aparece el comando que se ejecutaría en *TLC Console*.

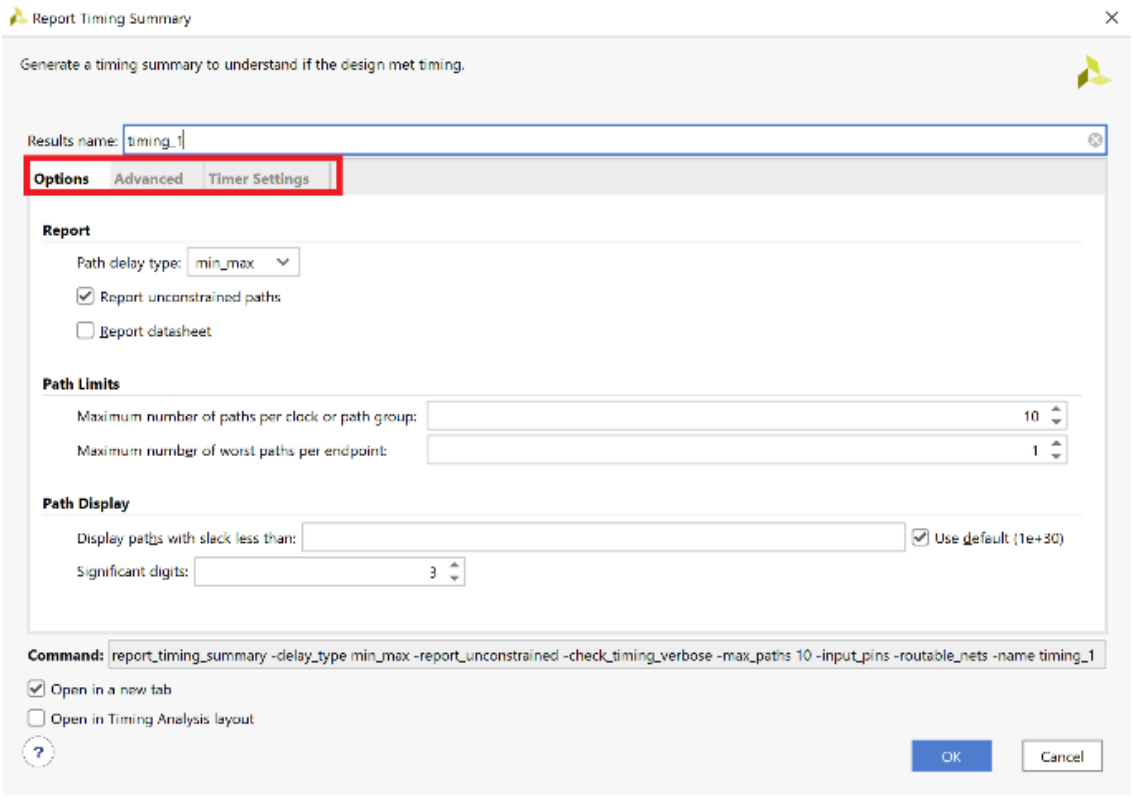


Figura 105: Vivado - Report Timing Summary – Parte 2

- *Options*

En esta pestaña se encuentra las secciones: *Report*, *Path Limits* y *Path Display* (Figura 106).

- *Report*

En esta sección se incluye *Path delay type*, que es el tipo de retardo en el camino. Este puede ser *min*, *max* o *min_max*. Según nuestra elección, se ejecutará un tipo de análisis. Por ejemplo, para comprobar el análisis de tiempos de *SETUP/RECOVERY* se selecciona *max*, para el análisis de tiempos de *HOLD/REMOVAL*, se selecciona *min*, y para comprobar ambos, se selecciona *max_min*.

También en esta sección se puede seleccionar la opción *Report unconstrained paths*, que informa de los caminos que no tienen requerimientos de tiempos. En Vivado esta opción está seleccionada por defecto.

Por último, para esta sección es la opción *Report Datasheet*, que genera el reporte el *datasheet* del sistema.

- *Path Limits*

Para establecer en el resultado del reporte un número máximo de rutas donde recoger el tiempo de ruta, se indica con número en la sección *Maximun number of paths per clock or path group*.

En el segundo apartado de esta sección se puede seleccionar el máximo número de peores rutas por fin de ruta, *Maximun number of worst paths per endpoint*.

- *Path Display*

En esta sección se puede configurar un límite de *Slack* máximo para los caminos mostrados, *Display paths with slack less than*.

Además, se puede indicar el número de dígitos significativos, *Significant digits*.

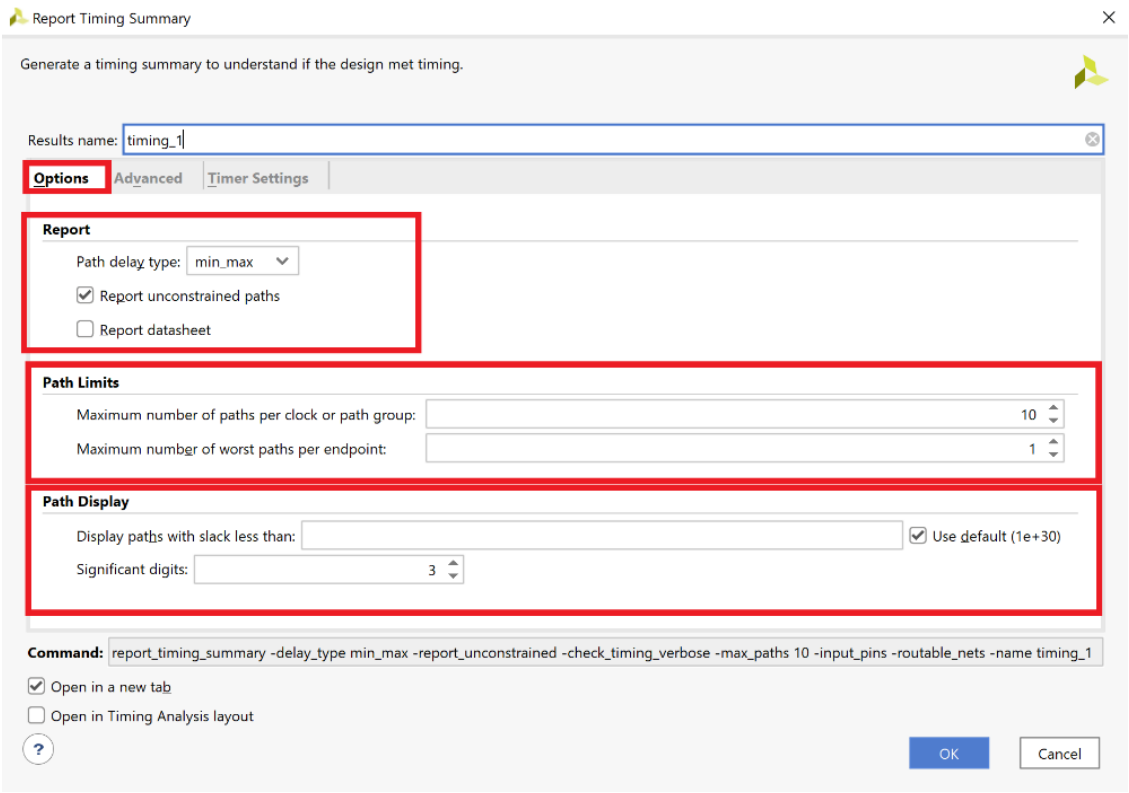


Figura 106: Vivado - Report Timing Summary – Parte 3

- *Advance*

En la configuración avanzada se encuentran las secciones: *Report*, *File Output* y *Miscellaneous* (Figura 107).

- *Report*

Esta sección ofrece la posibilidad de seleccionar que partes del circuito se quieren analizar. Se puede filtrar el análisis desde las celdas seleccionadas en *Report from cells* haciendo clic en los tres puntos para incluir qué celdas se quieren analizar (por defecto no seleccionado).

También se pueden mostrar los pines de entrada en la ruta analizada (por defecto seleccionado).

Otra opción es mostrar el número de conexiones de la ruta, *Report number of routable nets*.

Por último, se puede mostrar solo una ruta por cada set de pines únicos seleccionando *Report unique pins* (por defecto no seleccionado).

- *File Output*

En esta sección se pueden guardar los resultados en un archivo: *Export to file* e *Interactive report file*.

- *Miscellaneous*

Por un lado, se puede indicar que se ignoren los errores de comando sin que reporte ningún mensaje: *Ignore command errors*, o cancelar cualquier filtro de mensaje, para mostrar todos los mensajes: *Suspend message limits during command execution*.

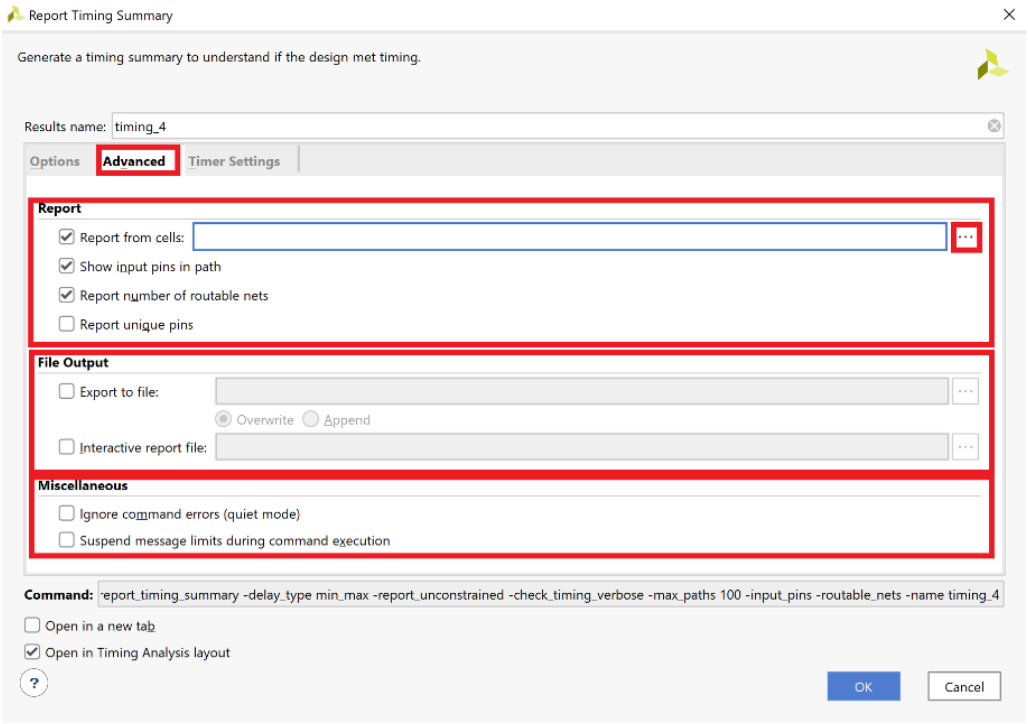


Figura 107: Vivado - Report Timing Summary – Parte 4

- *Timer Settings*

En la última pestaña para la configuración del reporte de tiempos se encuentran las secciones: *Interconnect*, *Speed grade*, *Multi-Corner Configuration* y *Disable flight delays* (Figura 108).

- *Interconnect*

  Esta sección comprende tres opciones: *actual*, *estimated* y *none*.

  - *Actual*: proporciona el retardo más preciso para las interconexiones del diseño (seleccionado por defecto).
  - *Estimated*: incluye una estimación de los retardos por interconexiones.
  - *None*: no incluye ningún retardo de interconexión en el análisis de tiempos, únicamente el retardo debido a la lógica del circuito es incluido.

- *Speed grade*

  Se puede realizar el análisis con el grado de velocidad por defecto en la FPGA (-1), o se puede seleccionar otro grado.

- *Multi-Corner Configuration*

  En esta sección se puede seleccionar el tipo de retardo establecido en el *Slow Corner* y el *Fast Corner*.

  - *Slow corner*: en este caso se comprueba el análisis del sistema cuando es más lento, para cada uno de los caminos de la ruta de datos.
  - *Fast corner*: por otro lado, con esta opción, se comprueba el análisis del sistema cuando es más rápido, para cada uno de los caminos de la ruta de datos.

  Ambos, se pueden configurar con *min*, *max* o *min_max* (por defecto) *delay*.

- *Disable flight delays*

  Esta opción se utiliza para deshabilitar los retardos de paquete al cálculo de retardos de I/O.

Una vez configurado el reporte, se debe hacer clic en *OK* para aplicar los cambios efectuados.

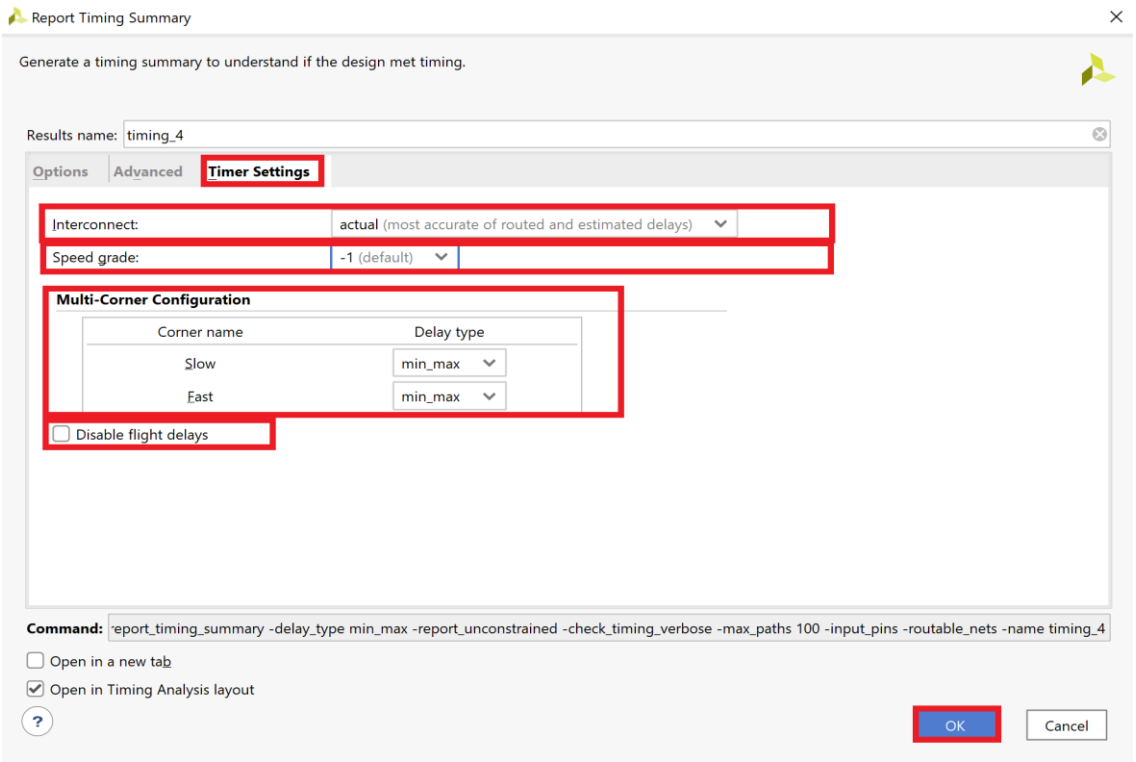


Figura 108: Vivado - Report Timing Summary – Parte 5

ANEXO 4.1. REPORT TIMING

Una vez se ha configurado el reporte de tiempos, en el apartado de resultados de Vivado aparecerá una nueva pestaña llamada *Timing* donde tendremos la información relativa a este reporte.

Este reporte se muestra la siguiente información:

- ***General Information***

En español, información general, provee información sobre los siguientes campos (Figura 109):

  - Nombre del diseño
  - Dispositivo seleccionado, paquete software y grado de velocidad de la placa (FPGA)
  - Versión de Vivado
  - Fecha actual del proyecto
  - Comando TCL equivalente

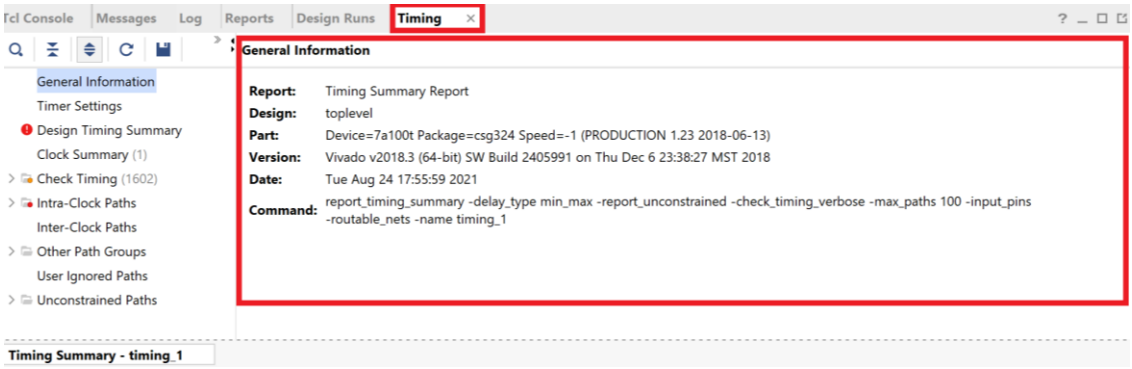


Figura 109: Vivado - Resultados reporte de tiempo - General information

- ***Timer Settings***

En esta sección aparecen los ajustes que se han configurado previamente en el reporte de tiempo, tal y como aparece en la Figura 110.

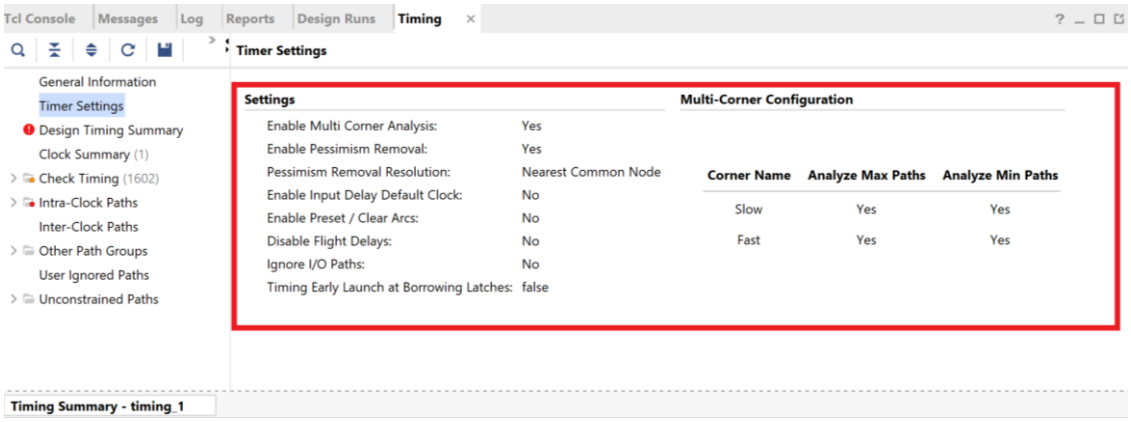


Figura 110: Vivado - Resultados reporte de tiempo - Timer Settings

- ***Design Timing Summary***

Esta es una sección de gran importancia en el estudio de mejoras en el diseño de un sistema, ya que presenta el resumen de los tiempos de los peores caminos y el tiempo total de Slack, lo que indica donde puede presentarse la posible mejora. Haciendo clic en el tiempo en azul, se dirige a la sección donde muestra la ruta del sistema que tiene ese tiempo. En la siguiente Figura 111 se presenta esta sección a modo de ejemplo.

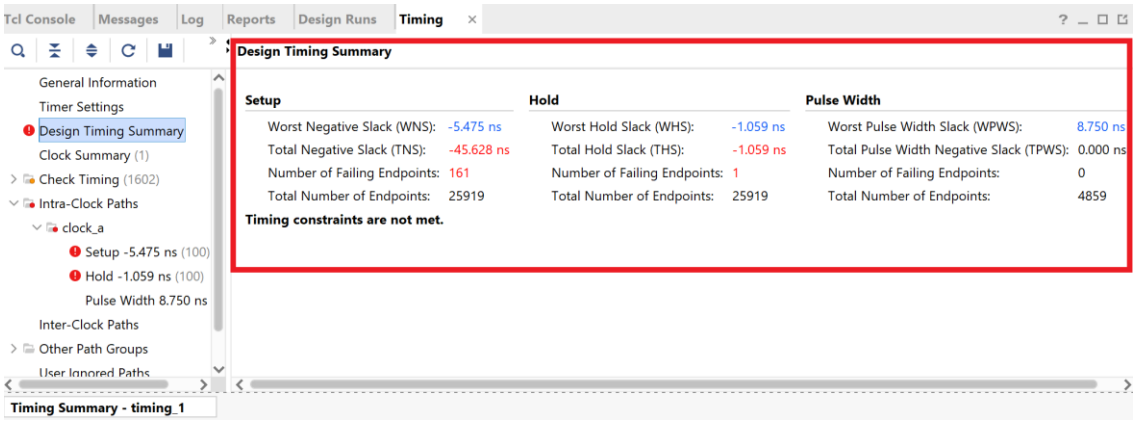


Figura 111: Vivado - Resultados reporte de tiempo - Design Timing Summary

- ***Clock Summary***

  En esta sección se incluye información sobre los relojes usados en el sistema.

- ***Check Timing***

  En esta sección se comprueba si faltan restricciones de tiempo o rutas con problemas de restricciones.

- ***Intra-Clock Paths***

  Esta sección se explicará más adelante con un poco más de detenimiento, ya que nos dará detalles de los tiempos que nos ofrece el reporte y se podrá utilizar para localizar las rutas que suponen un peor resultado a nivel de retardo del sistema.

- ***Inter-Clock Paths***

  Esta sección es similar a la anterior, pero entre diferentes rutas de diferentes relojes del sistema. En el caso de estudio que ocupa este trabajo, al existir un solo reloj, esta sección se podrá omitir.

- ***Other Path Groups***

  Aquí se muestran los grupos de rutas predefinidos y los grupos de ruta definidos por el usuario.

- ***User Ignored Paths***

  En esta sección se muestra las rutas que son ignoradas durante el análisis de tiempo, las cuales han sido previamente configuradas en las restricciones.

- ***Unconstrained Paths***

  En esta última sección aparecen las rutas que no han sido analizadas debido a que no se encuentran restricciones de tiempo para esa ruta.

Intra-Clock Paths

Como se ha comentado anteriormente, esta sección del reporte comparte información relevante para el estudio de los tiempos de ejecución del procesador. Aquí se mostrarán resúmenes de las estadísticas de tiempo entre el inicio y el final de ruta de un reloj del sistema, además de un detalle todas las rutas que se han configurado en el reporte.

Como se aprecia en la siguiente Figura 112 existen una serie de campos que dan información de cada ruta que aparece en el reporte.

- ***Slack***: es el tiempo de holgura que tiene un camino para que su resultado pueda permitirse un retardo, sin que afecte a otra parte del sistema.
- ***Levels***: es el número de componentes que atraviesan los datos en una ruta, excluyendo el del principio y el del final.
- ***Routes***: es el número de nodos entre componentes.
- ***High Fanout***: indica el número de nodos con alto "fanout" (número de entradas a puertas que es posible conectar). Si este número es alto, es posible que se produzcan violaciones de tiempo.
- ***From – To***: es el origen y el destino de la ruta en cuestión.
- ***Total Delay***: es la suma entre el retardo lógico y el retardo de nodo totales.
- ***Logic Delay***: es el retardo que producen los componentes en la ruta.
- ***Net Delay***: es el retardo que producen los nodos en la ruta.
- ***Requirement***: es el tiempo que requiere la ruta para que se cumplan las restricciones de tiempo.
- ***Source / Destination Clock***: es el reloj origen y destino de la ruta (en este estudio, será el mismo).
- ***Clock Uncertainly***: muestra un tiempo por la posible imprecisión del reloj debido al *jitter*, el retardo producido al variar una señal. Depende del *Jitter* total del sistema (TSJ, *Total System Jitter*), el jitter de entrada total del sistema (TIJ, *Total Input Jitter*), una variable del *Jitter* (DJ, *Discrete Jitter*) y el error de fase (PE, *Phase Error*). La fórmula de este tienpo es la siguienteEcuación 1: Clock Uncertainly:

$$Tclk\_uncertainly = \frac{\sqrt{(TSJ^2 + TIJ^2} + DJ}{2} + PE$$

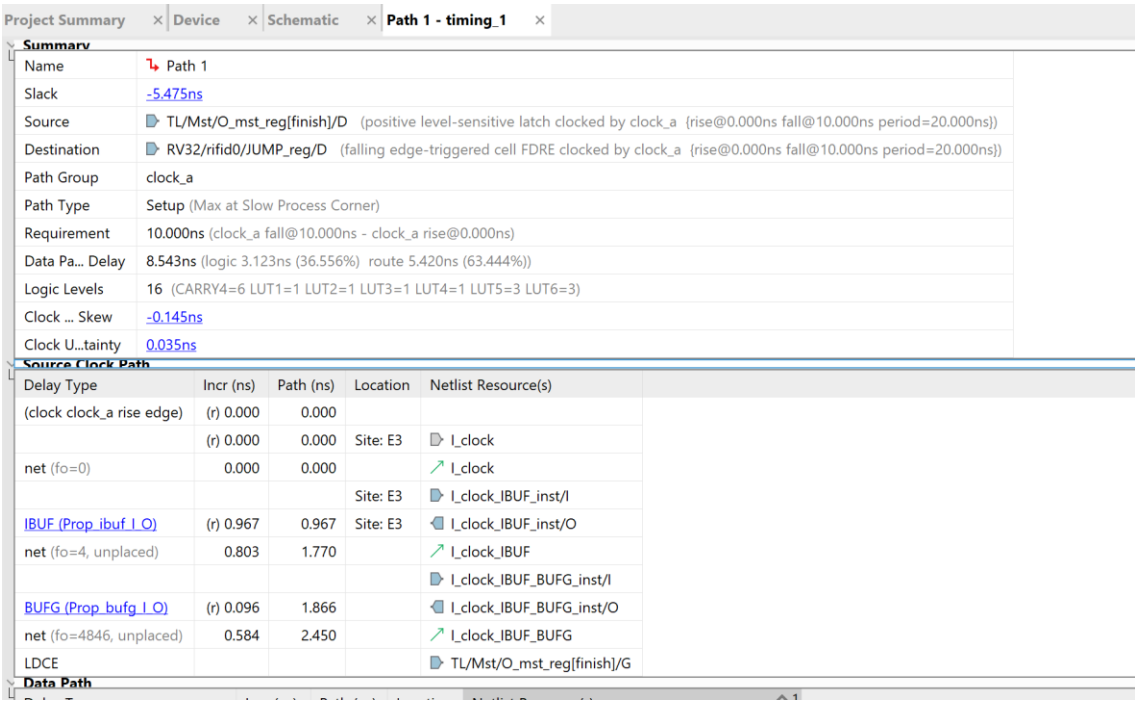
Ecuación 1: Clock Uncertainly

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Cl...	Clock ...
Path 66	-0.254	3	4	320	TL/Mst...ish]/D	RV32/re...[0]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 67	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...10]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 68	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...11]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 69	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...12]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 70	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...13]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 71	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...14]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 72	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...15]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035
Path 73	-0.254	3	4	320	TL/Mst...ish]/D	RV32/r...16]/CE	3.076	0.738	2.338	10.0	clock_a	clock_a	0.035

Figura 112: Vivado - Resultados reporte de tiempo - Rutas WCET

Si seleccionamos una ruta y hacemos doble clic en ella, se abrirá una sección en la parte de *Workspace* de Vivado, donde se verán los detalles de tiempo de la ruta. En la siguiente Figura 113 se puede observar que hay diferentes partes:

- Summary: detalles generales de la ruta seleccionada.
- Source clock path: detalles del origen de la ruta.
- Data clock path: detalles del camino que sigue la ruta.
- Destination clock path: detalles del destino de la ruta.



Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clock_a rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: E3	I_clock
net (fo=0)	0.000	0.000		I_clock
			Site: E3	I_clock_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 0.967	0.967	Site: E3	I_clock_IBUF_inst/O
net (fo=4, unplaced)	0.803	1.770		I_clock_IBUF
				I_clock_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_O)	(r) 0.096	1.866		I_clock_IBUF_BUFG_inst/O
net (fo=4846, unplaced)	0.584	2.450		I_clock_IBUF_BUFG
LDCE				TL/Mst/O_mst_reg[finish]/G

Figura 113: Vivado - Resultados reporte de tiempo - Detalles de la ruta seleccionada

En este apartado se puede comprobar con gran detalle todos los tiempos relativos a la ruta seleccionada, comprobando qué tipos de componentes tardan más en propagar el dato, o qué nodo ralentiza el paso del dato de un componente a otro.

ANEXO 5: CÓDIGO

En este anexo se presenta el código creado para el diseño del comparador, además del código de instrucciones utilizado para la realización de las pruebas (*Quicksort8_trace*).

- *to_std_logic.vhd*

Se ha desarrollado un paquete que engloba la función usada en el código del comparador. En esta función llamada *TO_STD_LOGIC*, se comprueba si una entrada es verdadera o falsa, es decir, si en la entrada se cumple una condición o no. En caso de que la entrada sea verdadera o *TRUE* (con un valor lógico de ´1´), la salida devolverá un valor lógico ´1´. En cambio, si la entrada es falsa o *FALSE* (con un valor lógico de ´0´), la salida devolverá un valor lógico ´0´. Es decir, lo que se consigue con esto, es que el comparador trabaje con valores *std_logic*, en vez de que trabaje con valores *std_logic_vector*, ya que lo que se pretende es activar un *flag* en caso de que se cumpla una condición. A continuación, se puede comprobar el código resultante creado para esta función.

LIBRARY

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.constants.all;
```

ENTITY

```
package comp_use is
    function to_std_logic(input : in boolean) return std_logic;
end package comp_use;
```

ARCHITECTURE

```
package body comp_use is
    function to_std_logic(input : in boolean) return std_logic is
    begin
            if input then
                    return '1';
            else
                    return '0';
            end if;
    end function to_std_logic;
end package body comp_use;
```

- comparator.vhd

Por otro lado, se ha programado un comparador que permitirá realizar las comparaciones en aquellas instrucciones que necesiten su uso, como es el caso de los saltos condicionales (*branchs*). Para ello, se ha creado un componente con tres entradas y una salida.

- ***func***: es una entrada de 3 bits que define el tipo de comparación que se va a realizar entre dos datos como, por ejemplo, si uno de ellos es igual, menor o mayor que el otro.
- ***op1 y op2***: son dos entradas de 32 bits, correspondientes a los operandos que van a ser comparados.
- ***result***: es la salida de 1 bit del comparador. Funciona como un *flag* que se enviará al sistema para decidir si se realiza el salto o no.

Se ha creado un proceso cuyas entradas son las entradas del componente, es decir, la función a realizar por el comparador y los operandos. La salida "*result*" dependerá de si el resultado de la comparación entre los operandos se corresponde con lo que especifica la funcionalidad que se está introduciendo en "*func*". Si es correcta, el resultado será ´1´ y el procesador realizará un salto a la instrucción indicada por la dirección calculada. En caso de no serlo, el resultado será ´0´ y el salto no se producirá, haciendo que el procesador siga el transcurso del programa.

LIBRARY

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library work;
use work.constants.all;
use work.comp_use.all;
```

ENTITY

```
entity comparator is
    port(
            func: in std_logic_vector(2 downto 0);
            op1, op2: in std_logic_vector(XLEN-1 downto 0);
            result: out std_logic
            );
end comparator;
```

ARCHITECTURE

```
architecture behavioural of comparator is
begin
  process(op1, op2, func)
  begin
        case func is
            when FUNC_BEQ =>
                result <= to_std_logic(op1 = op2);
            when FUNC_BNE =>
                result <= to_std_logic(op1 /= op2);
            when FUNC_BLT =>
                result <= to_std_logic(signed(op1) < signed(op2));
            when FUNC_BGE =>
                result <= to_std_logic(signed(op1) >= signed(op2));
            when FUNC_BLTU =>
                result <= to_std_logic(unsigned(op1) < unsigned(op2));
            when FUNC_BGEU =>
                result <= to_std_logic(unsigned(op1) >= unsigned(op2));
            when others =>
                result <= '0';
        end case;
    end process;
end behavioural;
```

- ´*Quicksort8_trace*

A continuación, se presenta el código en lenguaje ensamblador, que el SRG ha creado para realizar las pruebas del procesador.

```
00000000 <main>:
   0:   fc010113        addi    sp,sp,-64
   4:   02112e23        sw      ra,60(sp)
   8:   02812c23        sw      s0,56(sp)
   c:   04010413        addi    s0,sp,64
  10:   0000050b        0x50b
  14:   02700793        li      a5,39
  18:   fcf42223        sw      a5,-60(s0)
  1c:   00800793        li      a5,8
  20:   fcf42423        sw      a5,-56(s0)
  24:   05100793        li      a5,81
  28:   fcf42623        sw      a5,-52(s0)
  2c:   05900793        li      a5,89
  30:   fcf42823        sw      a5,-48(s0)
  34:   03800793        li      a5,56
  38:   fcf42a23        sw      a5,-44(s0)
  3c:   01000793        li      a5,16
  40:   fcf42c23        sw      a5,-40(s0)
  44:   03d00793        li      a5,61
  48:   fcf42e23        sw      a5,-36(s0)
  4c:   05700793        li      a5,87
  50:   fef42023        sw      a5,-32(s0)
  54:   00800793        li      a5,8
  58:   fef42223        sw      a5,-28(s0)
  5c:   fe442783        lw      a5,-28(s0)
  60:   fff78713        addi    a4,a5,-1
  64:   fc440793        addi    a5,s0,-60
  68:   00070613        mv      a2,a4
  6c:   00000593        li      a1,0
  70:   00078513        mv      a0,a5
  74:   214000ef        jal     ra,288 <quickSort>
  78:   fe042623        sw      zero,-20(s0)
  7c:   03c0006f        j       b8 <main+0xb8>
  80:   fec42783        lw      a5,-20(s0)
  84:   00279793        slli    a5,a5,0x2
  88:   ff040713        addi    a4,s0,-16
  8c:   00f707b3        add     a5,a4,a5
  90:   fd47a783        lw      a5,-44(a5)
  94:   00178713        addi    a4,a5,1
  98:   fec42783        lw      a5,-20(s0)
  9c:   00279793        slli    a5,a5,0x2
  a0:   ff040693        addi    a3,s0,-16
  a4:   00f687b3        add     a5,a3,a5
  a8:   fce7aa23        sw      a4,-44(a5)
  ac:   fec42783        lw      a5,-20(s0)
  b0:   00178793        addi    a5,a5,1
  b4:   fef42623        sw      a5,-20(s0)
  b8:   fec42703        lw      a4,-20(s0)
  bc:   fe442783        lw      a5,-28(s0)
  c0:   fcf740e3        blt     a4,a5,80 <main+0x80>
  c4:   fe042423        sw      zero,-24(s0)
  c8:   03c0006f        j       104 <main+0x104>
  cc:   fe842783        lw      a5,-24(s0)
  d0:   00279793        slli    a5,a5,0x2
  d4:   ff040713        addi    a4,s0,-16
  d8:   00f707b3        add     a5,a4,a5
  dc:   fd47a783        lw      a5,-44(a5)
  e0:   fff78713        addi    a4,a5,-1
  e4:   fe842783        lw      a5,-24(s0)
  e8:   00279793        slli    a5,a5,0x2
  ec:   ff040693        addi    a3,s0,-16
  f0:   00f687b3        add     a5,a3,a5
  f4:   fce7aa23        sw      a4,-44(a5)
  f8:   fe842783        lw      a5,-24(s0)
  fc:   00178793        addi    a5,a5,1
 100:   fef42423        sw      a5,-24(s0)
 104:   fe842703        lw      a4,-24(s0)
 108:   fe442783        lw      a5,-28(s0)
 10c:   fcf740e3        blt     a4,a5,cc <main+0xcc>
 110:   0000058b        0x58b
 114:   00000013        nop
 118:   03c12083        lw      ra,60(sp)
 11c:   03812403        lw      s0,56(sp)
 120:   04010113        addi    sp,sp,64
 124:   00008067        ret

00000128 <swap>:
 128:   fd010113        addi    sp,sp,-48
 12c:   02812623        sw      s0,44(sp)
 130:   03010413        addi    s0,sp,48
 134:   fca42e23        sw      a0,-36(s0)
 138:   fcb42c23        sw      a1,-40(s0)
 13c:   0000060b        0x60b
 140:   fdc42783        lw      a5,-36(s0)
 144:   0007a783        lw      a5,0(a5)
 148:   fef42623        sw      a5,-20(s0)
 14c:   fd842783        lw      a5,-40(s0)
 150:   0007a703        lw      a4,0(a5)
 154:   fdc42783        lw      a5,-36(s0)
 158:   00e7a023        sw      a4,0(a5)
 15c:   fd842783        lw      a5,-40(s0)
 160:   fec42703        lw      a4,-20(s0)
 164:   00e7a023        sw      a4,0(a5)
 168:   0000068b        0x68b
 16c:   00000013        nop
 170:   02c12403        lw      s0,44(sp)
 174:   03010113        addi    sp,sp,48
 178:   00008067        ret

0000017c <partition>:
 17c:   fd010113        addi    sp,sp,-48
 180:   02112623        sw      ra,44(sp)
 184:   02812423        sw      s0,40(sp)
 188:   03010413        addi    s0,sp,48
 18c:   fca42e23        sw      a0,-36(s0)
 190:   fcb42c23        sw      a1,-40(s0)
 194:   fcc42a23        sw      a2,-44(s0)
 198:   0000070b        0x70b
 19c:   fd442783        lw      a5,-44(s0)
 1a0:   00279793        slli    a5,a5,0x2
 1a4:   fdc42703        lw      a4,-36(s0)
 1a8:   00f707b3        add     a5,a4,a5
 1ac:   0007a783        lw      a5,0(a5)
 1b0:   fef42223        sw      a5,-28(s0)
 1b4:   fd842783        lw      a5,-40(s0)
 1b8:   fff78793        addi    a5,a5,-1
 1bc:   fef42623        sw      a5,-20(s0)
 1c0:   fd842783        lw      a5,-40(s0)
 1c4:   fef42423        sw      a5,-24(s0)
 1c8:   0640006f        j       22c <partition+0xb0>
 1cc:   fe842783        lw      a5,-24(s0)
 1d0:   00279793        slli    a5,a5,0x2
 1d4:   fdc42703        lw      a4,-36(s0)
 1d8:   00f707b3        add     a5,a4,a5
 1dc:   0007a783        lw      a5,0(a5)
 1e0:   fe442703        lw      a4,-28(s0)
 1e4:   02e7de63        bge     a5,a4,220 <partition+0xa4>
 1e8:   fec42783        lw      a5,-20(s0)
```

```
1ec:  00178793        addi    a5,a5,1
1f0:  fef42623        sw      a5,-20(s0)
1f4:  fec42783        lw      a5,-20(s0)
1f8:  00279793        slli    a5,a5,0x2
1fc:  fdc42703        lw      a4,-36(s0)
200:  00f706b3        add     a3,a4,a5
204:  fe842783        lw      a5,-24(s0)
208:  00279793        slli    a5,a5,0x2
20c:  fdc42703        lw      a4,-36(s0)
210:  00f707b3        add     a5,a4,a5
214:  00078593        mv      a1,a5
218:  00068513        mv      a0,a3
21c:  f0dff0ef        jal     ra,128 <swap>
220:  fe842783        lw      a5,-24(s0)
224:  00178793        addi    a5,a5,1
228:  fef42423        sw      a5,-24(s0)
22c:  fd442703        lw      a4,-44(s0)
230:  fe842783        lw      a5,-24(s0)
234:  f8e7cce3        blt     a5,a4,1cc <partition+0x50>
238:  fec42783        lw      a5,-20(s0)
23c:  00178793        addi    a5,a5,1
240:  00279793        slli    a5,a5,0x2
244:  fdc42703        lw      a4,-36(s0)
248:  00f706b3        add     a3,a4,a5
24c:  fd442783        lw      a5,-44(s0)
250:  00279793        slli    a5,a5,0x2
254:  fdc42703        lw      a4,-36(s0)
258:  00f707b3        add     a5,a4,a5
25c:  00078593        mv      a1,a5
260:  00068513        mv      a0,a3
264:  ec5ff0ef        jal     ra,128 <swap>
268:  0000078b        0x78b
26c:  fec42783        lw      a5,-20(s0)
270:  00178793        addi    a5,a5,1
274:  00078513        mv      a0,a5
278:  02c12083        lw      ra,44(sp)
27c:  02812403        lw      s0,40(sp)
280:  03010113        addi    sp,sp,48
284:  00008067        ret

00000288 <quickSort>:
288:  fd010113        addi    sp,sp,-48
28c:  02112623        sw      ra,44(sp)
290:  02812423        sw      s0,40(sp)
294:  03010413        addi    s0,sp,48
298:  fca42e23        sw      a0,-36(s0)
29c:  fcb42c23        sw      a1,-40(s0)
2a0:  fcc42a23        sw      a2,-44(s0)
2a4:  0000080b        0x80b
2a8:  fd842703        lw      a4,-40(s0)
2ac:  fd442783        lw      a5,-44(s0)
2b0:  04f75463        bge     a4,a5,2f8 <quickSort+0x70>
2b4:  fd442603        lw      a2,-44(s0)
2b8:  fd842583        lw      a1,-40(s0)
2bc:  fdc42503        lw      a0,-36(s0)
2c0:  ebdff0ef        jal     ra,17c <partition>
2c4:  fea42623        sw      a0,-20(s0)
2c8:  fec42783        lw      a5,-20(s0)
2cc:  fff78793        addi    a5,a5,-1
2d0:  00078613        mv      a2,a5
2d4:  fd842583        lw      a1,-40(s0)
2d8:  fdc42503        lw      a0,-36(s0)
2dc:  fadff0ef        jal     ra,288 <quickSort>
2e0:  fec42783        lw      a5,-20(s0)
2e4:  00178793        addi    a5,a5,1
```

BIBLIOGRAFÍA

[1] A. D. silva y A. M. H. P. P. E. S. S. P. Oscar Rodriguez Polo, «"UN MÉTODO Y UN DISPOSITIVO DE PROCESAMIENTO EN PARALELO DE INSTRUCCIONES DE PROGRAMA E INSTRUCCIONES DE TRAZA" - P201830266,» 2020. [En línea]. Available: https://www.patentes-y-marcas.com/patente/un-metodo-y-un-dispositivo-de-procesamiento-en-paralelo-de-instrucciones-de-programa-e-instrucciones-de-traza-p201830266.

[2] I. G. d. Río, «Modificación sobre una arquitectura RISC-V para poder procesar en paralelo instrucciones de traza y de programa,» Universidad de Alcalá, 2019.

[3] A. M. H. O. R. P. M. J. P. P. A. d. S. J. S. S. S. Iván Gamino del Rio, «A RISC-V Processor Design for Transparent Tracing,» Electronics, 2020. [En línea]. Available: https://www.mdpi.com/2079-9292/9/11/1873.

[4] D. Patterson y A. Waterman, «Guía Practica de RISC-V: el atlas de una Arquitectura Abierta,» 2017. [En línea]. Available: http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf.

[5] D. A. Patterson y J. L. Hennessy, «Computer Organization and Design. The Hadware/Software interface RISC-V Edition,» de *4. The Procesor*, pp. pp 473 - 735.

[6] Xilinx, «Vivado Design Suite User Guide. Design Analysis and Closure Techniques,» 2020. [En línea]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug906-vivado-design-analysis.pdf. [Último acceso: Julio 2021].

[7] Xilinx, «"Descarga Vivado",» [En línea]. Available: https://www.xilinx.com/support/download.html.. [Último acceso: 2021].

[8] Digilent, «Datasheet Nexys 4 DDR,» [En línea]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start. [Último acceso: 2021].

Universidad
de Alcalá